R. SCHULZ

60 SOLUTIONS POUR ORIC 1 + ATMOS

MATERIEL
EXTENSIONS
LOGICIELS

MICRO SYSTEMES

ETSF

Collection
MICRO-SYSTEMES

60 solutions pour Oric 1 et Atmos

COLLECTION E.T.S.F. MICRO-SYSTEMES

1 - A. VILLARD et M. MIAUX, *Un microprocesseur pas à pas*
2 - A. VILLARD ET M. MIAUX, *Systèmes à microprocesseur*
3 - P. GUEULLE, *Maîtrisez votre ZX 81*
4 - E. FLOEGEL, *Du Basic au Pascal*
5 - P. COURBIER, *Vous avez dit Basic ?*
6 - M. MARCHAND, *Vous avez dit micro ?*
7 - P. GUEULLE, *Pilotez votre ZX 81*
8 - M. JACQUELIN, *La micro-informatique et son A-B-C*
9 - M. OURY, *Maîtrisez les TO 7 et TO 7-70*
10 - P. GUEULLE, *Pilotez votre Oric 1 + Atmos*
11 - P. JOUVELOT et D. LE CONTE DES FLORIS, *Systèmes d'exploitation et logiciel de base des micro-ordinateurs*
12 - P. GUEULLE, *Robotisez votre ZX 81*
13 - M. CAUT, *J'apprends le Basic*
14 - C. MALOSSE, C. TASSET, P. PRUT, *La micro, c'est pas sorcier !*
15 - R. GREGOIRE, *Bus IEEE*
16 - M. OURY, *Maîtrisez le MO5*
17 - P. GUEULLE, *Votre ordinateur et la télématique*
18 - P. COURBIER, *Connaissez-vous Macintosh ?*
19 - M. ROUSSELET, *Graphismes en kits*
20 - S. ARQUIE, *Micro-informatique et P.M.E.*
21 - R. SCHULZ, *60 solutions pour Oric 1 et Atmos*
22 - H. HUNIC, *Listes et tableaux en Basic*
23 - P. GUEULLE, *Faites de l'argent avec votre micro.*

ISBN 2-85535-104-9 ISSN 0759-9498

REMY SCHULZ

60 solutions pour Oric 1 et Atmos

Diffusion
EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue, 75940 Paris Cedex 19



ARCHITECTURE DU SYSTEME

PROBLEMES MATERIELS DE L'ORMOS

INTERFACES

EXTENSIONS

PROGRAMME BASIC EN RAM

APPLICATIONS

LANGAGE ET ROUTINES MACHINE

ERREURS DU BASIC DE L'ORIC 1

PROBLEMES USUELS BASIC

IMPRIMANTE

MAGNETOPHONE

OPTIMISATION DES PROGRAMMES

ECRAN ET ROUTINES MACHINE

ANNEXES



NOTES PRELIMINAIRES

La majeure partie de cet ouvrage est valable aussi bien pour l'Oric 1 que pour l'Atmos. Pour éviter d'avoir à le répéter par trop souvent nous avons forgé le néologisme ORMOS, et nous n'emploierons les termes Oric et Atmos que dans les cas particuliers intéressant l'un ou l'autre modèle seulement.

Nous avons tenté dans la mesure du possible de donner des programmes et des applications valables pour les deux systèmes. Du fait de cette compatibilité, certaines routines peuvent être simplifiées pour être utilisées sur un seul appareil.

Nous avons divisé l'ouvrage en 60 sections répondant en principe chacune à une question ou un problème précis, mais certaines rubriques peuvent présenter des applications dépassant largement leur dénomination. Ces sections sont groupées par affinités en chapitres, des (*V.*) renvoient à d'autres sections contenant des indications complémentaires.

Nous avons cherché à donner un maximum d'informations nouvelles, et n'avons en conséquence fait qu'effleurer certains sujets par ailleurs très bien développés dans d'autres ouvrages, dont on trouvera la bibliographie en annexe.

Nous avons adopté pour les programmes en langage machine la syntaxe assembleur la plus communément admise, laquelle diffère de la syntaxe propre à l'ORMOS dans la désignation des grandeurs hexadécimales. Nous avons cependant employé le # précédant une valeur hexadécimale dans tous les autres cas.

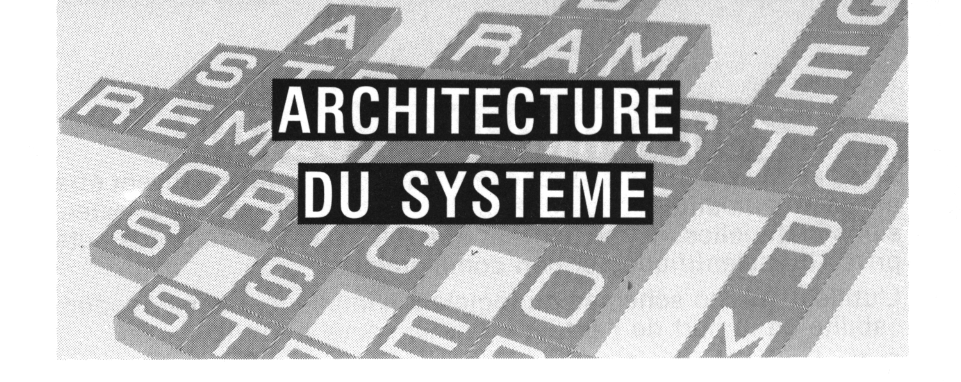
Le poussoir RESET de l'ORMOS déclenche en fait une interruption NMI. Nous emploierons cependant le mot RESET pour désigner l'appui sur ce poussoir, et nous aurons recours à l'expression « vrai RESET » pour désigner la réinitialisation du 6502.

Nous n'emploierons jamais la lettre O dans le nom des variables, pour éviter toute confusion avec le chiffre 0.

Si les astuces logicielles proposées dans cet ouvrage peuvent être utilisées et modifiées à toutes fins, il n'en va pas de même des extensions matérielles, qui ne peuvent être réalisées que dans des buts privés ou scientifiques et non commerciaux.

L'utilisation des schémas ou logiciels n'implique aucune responsabilité de la part de l'auteur ou de la société éditrice.

Enfin, le moi étant haïssable (mais tous les hommes ne sont-ils pas « ego » ?), l'auteur a employé la première personne du pluriel alors qu'il ne parle bien souvent qu'en son seul nom. Que lui soit pardonné ce travers princier.



L'Oric et l'Atmos possèdent presque exactement la même structure matérielle. Le circuit imprimé des deux appareils est identique, il porte les mêmes composants, les connecteurs fournissent les mêmes signaux. La seule différence notable est le clavier, lequel comporte une touche supplémentaire sur l'Atmos et dont l'aspect (l'aspect seul malheureusement) est plus professionnel que celui de l'Oric.

La différence essentielle est logicielle. La ROM ou mémoire morte ne contient pas les mêmes informations. Cela pourrait suffire à transformer radicalement le système, ce n'est en fait pas le cas, l'Oric et l'Atmos sont très largement compatibles. Nous décrirons leurs principales divergences plus loin, mais pour l'instant c'est donc de l'ORMOS qu'il s'agit, puisque les deux appareils peuvent être considérés comme matériellement identiques.

1 Boîtiers principaux

Commençons par une brève présentation des composants les plus importants ; leurs brochages se trouvent en annexe.

Au cœur du système se trouve le 6502, microprocesseur 8 bits capable d'adresser 64 ko de mémoire au moyen de 16 lignes d'adresses.

Au 6502 sont associés deux autres composants logiques essentiels, d'une grande complexité, l'ULA et le VIA 6522.

ULA, cela veut dire Uncommitted Logic Array, ce qui ne signifie pas grand-chose. Ce genre de composant se retrouve dans la plupart des systèmes de ce type. Ses fonctions sont étroitement associées à celles du microprocesseur, qu'il décharge de certaines tâches. Ici l'ULA assure notamment des fonctions d'horloge, de décodage d'adresses pour activer les circuits voulus, de rafraîchissement des RAM, et l'essentiel de l'interface vidéo.

Le VIA, ou Versatile Interface Adapter, est un boîtier que l'on retrouve fréquemment associé au 6502, un grand spécialiste des entrées-sorties. Et versatile, il l'est, notre VIA, puisque par lui transitent les données et les signaux de commande du clavier, du magnétophone, de l'imprimante et du synthétiseur sonore. Et son rôle ne se limite pas à cela.

Nous avons ensuite les huit boîtiers de RAM, chacun d'entre eux correspondant à un bit de données. Il s'agit de 4164, des circuits très récents contenant chacun 64 K bits adressables par huit entrées multiplexées. Seuls 48 K de cette RAM sont normalement utilisables par le système. En effet les 16 autres K correspondent à l'espace adressable de la ROM. Cette ROM est également un des plus récents produits de la technologie de l'intégration, 128 K bits sur une seule puce.

Nous ne nous préoccuperons guère dans cet ouvrage du synthétiseur sonore, le AY-3-8912. C'est également un circuit très complexe dont les possibilités sont largement sous-utilisées par le Basic de l'ORMOS. Nous nous contenterons de signaler que ce circuit possède un buffer (tampon) qui est employé par le port A du VIA pour la scrutation du clavier.

2 Fonctionnement du système

Après ce bref survol des principaux éléments de l'ORMOS nous pouvons nous pencher sur leur interdépendance. Un micro-système s'architecture autour de trois bus :

— le bus de données sur lequel circulent les informations,
— le bus d'adresses qui permet de déterminer dans quelle case mémoire sera lue ou écrite une donnée,
— le bus de commandes ; ce sont ici des signaux annexes qui contrôlent le fonctionnement du système.

Revenons maintenant au 6502. Sur ses 40 broches nous trouvons 8 lignes DB0 à DB7 correspondant au bus de données, 16 lignes AB0 à AB15 correspondant au bus d'adresses, et divers signaux. Ce sont essentiellement :

— des signaux d'horloge : tout le système bat au rythme d'une horloge à 1 MHz. Le 6502 reçoit sur sa broche PHI0 un signal d'horloge externe (ici il provient de l'ULA) à partir duquel il génère deux signaux rectangulaires en opposition de phase, PHI1 et PHI2,
— trois entrées d'interruptions NMI, RES et IRQ,

— une sortie R/W (Read/Write) : le 6502 signale par l'état logique de cette sortie s'il lit ou écrit sur le bus bidirectionnel de données. R/W est à l'état haut en lecture, bas en écriture.

Nous reviendrons sur ces différents signaux. Voyons maintenant grâce à la figure 1 comment les principaux éléments du système sont placés sur les bus d'adresses et de données. Pour conserver une certaine clarté au schéma, les signaux de commande n'ont pas été représentés.

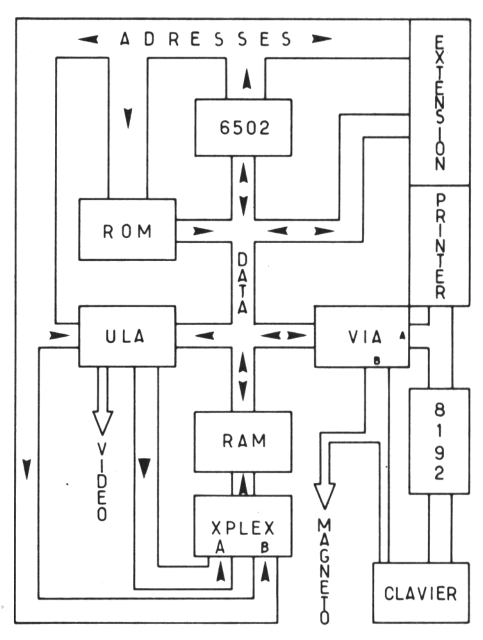


Fig. 1 - Architecture du système autour des bus d'adresses et de données. Le sens de circulation des informations est indiqué par > et <. Le bus de commandes n'est pas représenté.

Nous constatons que si le 6502 a directement accès par son bus d'adresses à la ROM, il n'en va pas de même pour la RAM. Le bus d'adresses se divise en deux. Les 8 bits de poids faible AB0 à AB7 vont vers les entrées B des circuits multiplexeurs commandant le décodage de la RAM, deux 74 LS 257, mais les 8 bits de poids fort AB8 à AB15 vont vers l'ULA. 8 lignes en ressortent vers les entrées A des multiplexeurs, nous les nommerons UB8 à UB15.

Il nous faut revenir sur les mémoires dynamiques 4164. Dans un

souci de compacité, ces boîtiers de 16 broches ne disposent que de 8 entrées d'adresses. Ces entrées reçoivent tour à tour les signaux de décodage de la rangée d'une case mémoire donnée, puis de sa colonne, validés par des impulsions sur les entrées RAS et CAS (Row et Column Adress Strobe). Ces 8 entrées sont commandées par les sorties des 74 LS 257, deux boîtiers possédant chacun deux fois 4 entrées, A et B, validées en sortie selon l'état de l'entrée SELECT. Cette broche est contrôlée par l'ULA, qui gère également les signaux RAS et CAS.

Résumons-nous. Les 8 adresses de poids faible AB0 à AB7 commandent donc les 8 entrées colonne de la RAM tandis que les 8 adresses de poids fort AB8 à AB15 transitent par l'ULA, qui génère 8 lignes correspondantes vers les entrées rangée de la RAM, UB8 à UB15. C'est encore l'ULA qui fournit les signaux validant ces adresses.

Mais l'ULA va assurer un décodage des bits d'adresses qu'elle reçoit. Imaginons de diviser l'espace adressable de 2^16 octets en 256 pages de 256 octets chacune, numérotées de 0 à 255. Ce sont les bits d'adresses de poids fort qui déterminent le numéro de la page (rangée) tandis que les bits de poids faible déterminent l'emplacement de l'octet choisi dans cette page (colonne). Si les 8 bits de poids fort ont une valeur décimale supérieure à 191, c'est-à-dire pour les adresses hexa de # C000 à # FFFF correspondant à la ROM, l'ULA portera au niveau bas sa broche 23, activant ainsi la ROM par son entrée CS (Chip Select). Par ailleurs elle ne fournira pas les impulsions RAS et CAS, ainsi les sorties de la RAM resteront à l'état de haute impédance et seule la ROM se trouvera sur le bus de données.

D'autre part, lorsque c'est la page 3 qui est adressée, c'est-à-dire pour les adresses de # 0300 à # 03FF, l'ULA ne fournit pas non plus les signaux de décodage de la RAM. Elle met à 0 en revanche sa broche 25, ce qui active le VIA par son entrée CS2. Le VIA se trouve alors seul sur le bus. Il possède 16 registres qui sont vus par le 6502 comme autant de cases mémoire, ces registres étant décodés par les bits d'adresses AB0 à AB3. Dans la configuration initiale du système les bits AB4 à AB7 ne sont pas décodés, en conséquence le VIA occupe toute la page 3. Il est cependant possible et en fait fort pratique d'effectuer un décodage partiel ou complet de ces adresses pour activer d'autres circuits d'entrées-sorties F4 *(V. 14).*

Pour toutes les autres adresses, c'est la RAM qui se trouve sur le bus, en lecture ou en écriture.

Est-ce si simple ? Non, bien sûr. L'ULA doit aussi lire la RAM pour

son propre compte, et ses exigences ne sont pas du tout les mêmes que celles du 6502. Elle balayera en fait seulement les emplacements mémoire qui l'intéressent, c'est-à-dire les adresses correspondant à l'écran, TEXT ou HIRES, les tables de caractères, et quelques adresses stratégiques en page 2. Comment cela va-t-il se faire ? Nous avons vu que l'ensemble du système battait au rythme d'une horloge qui synchronisait toutes les opérations, le signal PHI2. Lorsque PHI2 est au niveau haut, c'est le 6502 qui a seul accès au bus de données ; lorsqu'il est au niveau bas, c'est alors l'ULA qui lit seule les données présentes sur ce bus. Remarquons que l'ULA ne dispose que de 8 lignes d'adresses pour décoder la RAM : elle devra donc fournir successivement sur ces 8 lignes les parties haute et basse de l'adresse voulue. Notons aussi que si l'horloge bat à 1 MHz, ce qui représente une valeur faible pour un micro, certains circuits effectueront au moins quatre opérations pendant un cycle. Il ne faut pas oublier non plus que l'ULA doit aussi rafraîchir les RAM dynamiques, c'est-à-dire que chaque entrée colonne doit être portée au niveau haut au moins une fois par milliseconde.

Nous devons souligner que tout ce que nous venons de dire de l'ULA est hypothétique, Oric n'ayant guère donné de renseignements sur ce composant. Le fonctionnement du système tel que nous l'avons présenté est vraisemblablement simplifié, mais nos suppositions sont néanmoins assez proches de la réalité pour nous avoir permis de réaliser certaines extensions au système.

3 Le VIA 6522

Il n'est pas question ici de présenter toutes les possibilités de ce composant bien décrit par ailleurs *(V. bibliographie, « Entrées-sorties »).*

C'est avant tout un coupleur parallèle permettant l'interfaçage avec le monde extérieur, les données fluctuantes présentes sur le bus étant en effet dans la plupart des cas inutilisables par tout ce qui ne bat pas au même rythme que le système. Le 6522 possède deux ports parallèles de 8 bits, analogues à des verrous (latches), sur lesquels on peut soit stocker des octets de manière permanente, soit lire une information extérieure.

Le VIA occupe 16 adresses dans l'ORMOS, # 300 à # 30F (768 à 783). En fait, le décodage d'adresses étant incomplet dans la configuration de base, toutes les adresses # 3X0 à # 3XF sont équivalentes. Les réfractaires à l'hexa peuvent ainsi mémoriser l'emplacement du

VIA de 800 à 815. Mais si l'on réalise des extensions, il faudra obligatoirement laisser ce VIA-ci aux adresses #300 à #30F car le système ne peut y accéder que par ces seules valeurs.

Si l'on veut vérifier cette assertion en entrant une commande du genre :

FOR A = #300 TO #3FF : ? PEEK(A) ; : NEXT

on découvre que si certaines valeurs restent effectivement constantes, d'autres par contre varient. La raison en est que le VIA possède aussi deux timers, décrémentés au rythme de l'horloge à laquelle le VIA est également relié (timer 1 en #304-#305, timer 2 en #308-#309).

Le VIA possède donc deux ports, appelés port B et port A, occupant les registres 0 et 1, soit les adresses #300 et #301. Ces deux ports se matérialisent sur le boîtier du 6522 par deux fois dix broches nommées PBO à PB7, CB1 et CB2 d'une part, PA0 à PA7, CA1 et CA2 d'autre part. Les broches PB0 à PB7 et PA0 à PA7 correspondent aux bits de données de mêmes poids, elles peuvent être programmées bit par bit en entrées ou en sorties par deux registres de direction, DDRB et DDRA (Data Direction Register), occupant respectivement les adresses #302 et #303. Un bit à 0 dans ces registres signifie que la broche correspondante est une entrée, le bit à 1 signifiant bien sûr une sortie.

CB1 et CB2 d'une part, CA1 et CA2 d'autre part, sont des signaux de contrôle destinés à établir le dialogue avec le périphérique. Illustrons leur fonctionnement avec un mode de communication très classique dit en « Poignée de main », en sortie par exemple sur le port A.

La première opération consistera à configurer le VIA c'est-à-dire à établir le port A en sortie en chargeant dans le DDRA #FF (255 ou 11111111 en binaire) et à déterminer le rôle de CA1 et CA2 en chargeant d'autre registres. Généralement CA1 est une entrée et CA2 une sortie. Ensuite le 6502 ira chercher une par une les données à transmettre au périphérique et les chargera dans le registre 1 (port A). L'adressage de ce registre en mode « poignée de main » va provoquer plusieurs opérations :

— le bit 1 du registre 13 sera forcé à 0,
— la sortie IRQ du VIA sera réinitialisée,
— une impulsion négative sera transmise sur CA2, signalant au périphérique que la donnée est prête sur le port A.

Le programme attendra ensuite en principe un signal en provenance du périphérique signifiant que la donnée est enregistrée et qu'il est

prêt à en recevoir une autre. Ceci se fait souvent par une transition sur CA1, laquelle va positionner à 1 le bit 1 du registre 13 et éventuellement déclencher une interruption. Le 6502 ira alors chercher une nouvelle donnée qu'il chargera sur le port A...

Ceci n'est qu'un exemple un peu simplifié du fonctionnement du VIA qui peut avoir bien d'autres applications qu'il serait vain d'espérer décrire en quelques pages.

L'ORMOS utilise le port A pour communiquer avec l'imprimante, d'une manière tout à fait similaire à ce schéma, à ceci près que l'impulsion de sortie vers l'imprimante (STROBE) n'est pas délivrée sur CA2, mais sur un bit du port B, le PB4.

Les assignations complètes sont les suivantes :

— PORT A : tout en sortie (DDRA = # FF ou 11111111) :
  - PA0 à PA7 ont plusieurs fonctions : accès au synthétiseur sonore, scrutation des colonnes du clavier (via le buffer du synthétiseur sonore), port parallèle pour l'imprimante,
  - CA1 reçoit le signal ACK de l'imprimante,
  - CA2 sert à la sélection du synthétiseur (avec CB2),

— PORT B : tout en sortie sauf PB3 (DDRB = # F7 ou 11110111) :
  - PB0 à PB3 : scrutation des rangées du clavier,
  - PB4 envoie le STROBE vers l'imprimante,
  - PB5 est inutilisé par le système,
  - PB6 commande le relais de télécommande du magnétophone,
  - PB7 envoie les données sérielles vers le magnétophone,
  - CB1 reçoit les données du magnétophone,
  - CB2 sert à la sélection du synthétiseur (avec CA2).

Nous constatons qu'il se passe beaucoup de choses sur le port A, trop peut-être, nous y reviendrons.

A l'extérieur de l'ORMOS nous avons accès aux signaux suivants :

- PB4, PA0 à PA7, CA1 sur le connecteur de l'imprimante,
- PB7 et CB1 mis en forme sur le connecteur du cassette,
- Nous pouvons également utiliser le relais commandé par PB6.

Ces assignations ne sont pas définitives. Le programmeur peut décider de configurer le VIA à son idée, pourvu qu'il respecte certaines conditions. La première est une connaissance approfondie du composant et de ses possibilités, il ne faut pas poker n'importe quoi dans le VIA *(V.9)*. Il faudra également écrire un programme correspondant à ce que l'on désire, presque obligatoirement en langage machine. Il faudra enfin empêcher le système d'utiliser lui-même le VIA, sous peine de voir les contenus de ses registres rétablis par le Basic.

Ceci nous mène tout naturellement à la section suivante traitant des interruptions, mais essayons auparavant de mieux comprendre le fonctionnement du VIA à l'aide d'une application simple, en simulant l'instruction LPRINT en Basic. Voici tout de suite la routine :

```
9 REM Simulation de LPRINT A$
10 INPUT A$
20 A$=A$+CHR$(13)+CHR$(10)
30 POKE 803,255:POKE 812,221:POKE 814,64
40 FOR A=1 TO LEN(A$)
50 B=ASC(MID$(A$,A))
60 POKE 801,B
70 POKE 800,PEEK(800) AND 239:POKE 800,PEEK(800) OR 16
80 IF (PEEK(813) AND 2)<>2 THEN 80
90 NEXT
100 POKE 814,192
```

On veut donc imprimer A$. A la ligne 30 on configure le VIA. Les deux premiers POKE sont en fait inutiles, ils ne font que charger des valeurs déjà présentes dans ces deux registres. Le POKE en 814 interdit au VIA d'émettre des demandes d'interruption. Chaque caractère est ensuite chargé tour à tour dans le registre du port A. Puis on envoie l'impulsion STROBE en mettant à 0 puis à 1 le PB4 de la ligne 70. On attend ensuite la réponse de l'imprimante en 80. Une transition positive sur CA1 va en effet mettre à 1 le bit 1 du registre 13. En 100 on rétablit la valeur normale du registre 14 autorisant les interruptions.

Et que se passe-t-il, si l'on fait tourner ce programme, pourvu que l'on connecte une imprimante ? On va bien imprimer A$, mais il risque de se passer ensuite des choses inattendues. C'est le Basic qui fait des siennes. Il serait sans doute possible de résoudre ce problème, mais nous ne nous y sommes guère attardés. On programmera les routines d'entrées-sorties dans pratiquement tous les cas en langage machine, pour éviter des conflits avec le Basic d'une part, et d'autre part pour des raisons de rapidité d'exécution. La ligne 80 par exemple est inutile si l'imprimante possède un buffer d'entrée, car la donnée aura été enregistrée avant que cette ligne soit atteinte. Voyons maintenant ce qui se passe si l'on supprime le POKE 814,64 en ligne 30. On va bien imprimer quelque chose, mais pas du tout

ce que l'on désirait. Les données chargées sur le port A ont été perdues car ce port est utilisé par la routine de scrutation du clavier appelée à chaque interruption. Cet inconvénient évident en Basic subsiste néanmoins lorsque l'on utilise les instructions LPRINT et LLIST, principalement sur l'Oric. On aura un certain nombre d'erreurs, dépendant de la taille du buffer de l'imprimante et de la fréquence des interruptions. La meilleure méthode pour éliminer tout risque d'erreur est d'interdire les interruptions.

4 Les interruptions

Le 6502 possède trois entrées d'interruptions, NMI, RES et IRQ. Un niveau bas sur l'une de ces entrées a pour effet de brancher le microprocesseur à l'adresse stockée en #FFFA-#FFFB pour NMI, #FFFC-#FFFD pour RES, #FFFE-#FFFF pour IRQ. Ces trois couples d'adresses sont les fameux vecteurs du 6502, ils imposent que la mémoire morte d'un système basé sur lui ait au moins un bloc dans cette zone.

Ces trois entrées ont évidemment des rôles différents :

— RES signifie RESET, cela n'a aucun rapport avec le poussoir honteusement dissimulé sous l'ORMOS. C'est à l'adresse contenue en #FFFC-#FFFD que se branche le 6502 lorsqu'il est alimenté. C'est donc généralement le point d'entrée des routines d'initialisation totale du système ;

— IRQ signifie Interrupt ReQuest. C'est une demande d'interruption qui ne sera validée que si le bit de masque d'interruptions du registre d'état P du 6502 est à 0. Une instruction BRK va aussi brancher le 6502 en (#FFFE-#FFFF) ;

— NMI signifie Not Maskable Interruption. Un niveau bas sur NMI sera prioritaire et branchera obligatoirement le 6502 en (#FFFA-#FFFB) quel que soit l'état du bit de masque d'interruptions. L'inaccessible poussoir RESET déclenche en fait une interruption NMI.

Il y a en outre des points communs entre ces deux types d'interruptions. Dans les deux cas le microprocesseur ne se branchera sur le contenu des vecteurs qu'après avoir terminé l'instruction en cours et empilé les deux octets de l'adresse de l'instruction suivante et le registre d'état. Le retour d'interruptions se fera par la même instruction RTI. En outre sur l'ORMOS le contenu des vecteurs d'interruption nous envoie à des adresses en page 2, où l'on trouve des sauts renvoyant à des routines en ROM. Ce détour en RAM permet

au programmeur de détourner les interrruptions à son plus grand profit.

Dans la configuration initiale de l'ORMOS, les seules interruptions systématiques sont des interruptions IRQ demandées par le VIA. Comment cela se produit-il ? A l'initialisation le timer 1 du VIA et son verrou sont chargés avec la valeur #2710 (10 000). Cette valeur va être décrémentée à chaque impulsion d'horloge. Le registre d'autorisation d'interruption valide les interruptions déclenchées par le timer 1. Après 10 000 impulsions d'horloge ce timer sera donc à 0, et la sortie IRQ du VIA sera portée au niveau bas. Le 6502 interrompra alors sa tâche et procédera à diverses opérations, pourvu que le bit de masque d'interruptions soit à 0 et que la routine IRQ n'ait pas été détournée :

— il sauvegardera le contenu de ses registres (A, X, Y),
— il rechargera le timer 1 du VIA avec la valeur contenue dans le verrou en #306-#307 (#2710 en principe),
— il décrémentera les trois timers en #272-#273, #274-#275 et #276-#277, et procédera aux opérations voulues en cas de nullité,
— il regardera ce qui se passe du côté du clavier et appellera les routines concernées s'il y a lieu (BREAK pour CTRL C par exemple),
— il récupérera le contenu de ses registres et poursuivra son programme initial, jusqu'à la prochaine interruption.

Cela fait pas mal d'opérations, un nombre minimal de cycles d'horloge est requis pour chaque interruption. 1 MHz divisé par 10 000 donne 100 interruptions par seconde, ce qui fait beaucoup de cycles perdus. De fait nous conseillons d'interdire les interruptions pour trois raisons :

— la non-observation de cette règle entraîne des erreurs de lecture sur le port parallèle,
— l'ORMOS est suffisamment lent pour ne pas avoir besoin d'être interrompu cent fois par seconde. Le gain de temps est d'environ un tiers sur un programme non interrompu,
— beaucoup de pannes de l'Oric sont dues à la détérioration du VIA.

  Ne surchargeons donc pas de travail ce composant, si on lui souhaite une vieillesse heureuse.

Il est bien sûr possible de diminuer (ou d'accélérer) la fréquence des interruptions en dokant des valeurs supérieures (ou inférieures) à 10 000 en #306 ; nous ne le conseillons pas, en raison du dernier point.

De toute manière, dès que l'on n'utilise plus le VIA de manière absolument réglementaire, il est impératif d'inhiber les interruptions.

Un problème se pose alors : l'usager n'a plus la main et ne peut plus arrêter ses programmes que par un acrobatique appui sur le RESET qui ne permettra pas de relancer le programme dans bien des cas. Nous y apporterons des solutions logicielles *(V. 29, 43).*

PROBLEMES MATERIELS DE L'ORMOS

Considérons d'abord un point important avant d'aborder ces problèmes. Des pannes ou des défauts de l'ORMOS peuvent nécessiter une intervention souvent bénigne à l'intérieur de l'appareil. En principe l'ouverture du boîtier va vous faire perdre la garantie, à moins que votre intervention n'ait été particulièrement discrète. Il faut songer à l'achat de l'appareil que certains revendeurs n'effectuent aucune réparation, même enfantine, sur leurs produits, et qu'en cas de pépin, même minime, ils retourneront votre appareil à la maison mère, vous en privant pendant plusieurs semaines. Un revendeur agréé pour intervenir lui-même sur les machines pourra par contre vous dépanner rapidement, voire immédiatement dans les cas les plus simples.

L'ORMOS est un appareil remarquablement bien conçu, et remarquablement bon marché. Ces deux choses ne vont pas toujours de pair et, en fait, l'ORMOS souffre quelque peu de la compression des coûts de fabrication. Il est néanmoins facile d'apporter des remèdes à certains ennuis courants.

5 Le démarrage

C'est bien sûr le premier problème. Nous avons vu que le 6502, lorsqu'il recevait sa tension d'alimentation, se branchait à l'adresse contenue en # FFFC-# FFFD. Il peut arriver que ce branchement ne se produise pas, auquel cas le système ne peut s'initialiser. Les appareils professionnels disposent généralement d'un trigger expédiant à l'allumage une impulsion vers l'entrée RESET. Ce n'est pas le cas de l'ORMOS dont le comportement est parfois déroutant. L'initialisation dure en principe environ 3 secondes mais il arrive qu'elle se produise quasi instantanément, ce qui s'explique mal lorsque l'on sait que toutes les cases mémoire sont testées pendant cette ini-

tialisation. Il arrive malheureusement aussi qu'elle ne se produise pas, et que l'on branche et débranche rageusement la prise d'alimentation à de multiples reprises avant d'obtenir un résultat. Ces manœuvres ne sont pas conseillées, il vaut bien mieux faire un vrai RESET en portant à la masse la broche 4 du connecteur d'extension. Ceux qui ont des problèmes fréquents de démarrage pourront se monter un starter en installant un poussoir à contact fugitif établissant cette liaison.

6 Le RESET

Il s'agit du poussoir placé sous l'ORMOS qui n'a rien à voir avec le vrai RESET dont nous venons de parler. En voilà un que l'on ne risque guère de déclencher par inadvertance ! Son emploi est pourtant souvent nécessaire. Le montage d'un poussoir plus commodément placé s'avère vite utile, mais le signal NMI n'est pas disponible sur les connecteurs arrière. Il faudra donc ouvrir l'appareil pour le récupérer. Adieu la garantie... Il s'agit encore d'établir une liaison entre la broche NMI et la masse. Pendant que vous y êtes, pourquoi ne pas monter trois poussoirs NMI, RES et IRQ, qui pourront chacun trouver leur utilité ? Il existe une étroite bande, au-dessus du circuit imprimé du clavier, où l'on peut monter des boutons-poussoirs ou de petits inverseurs en perçant la face avant. Il faut s'assurer que ces composants ne gêneront pas la repose du circuit principal. Si l'on ne veut pas perdre la garantie, il faut recourir au système D (percer un trou sous la table par exemple) ou monter le SDEC 007 *(V. 18).*

7 L'alimentation

Il faut savoir que le régulateur intégré placé à l'intérieur de l'ORMOS est un régulateur négatif de type 7905. En conséquence le point commun entre l'alimentation extérieure et le circuit de l'ORMOS est le + 9 volts. La masse de l'ORMOS a un potentiel supérieur d'environ 4 volts à celle du câble d'alimentation. Il faut y porter la plus grande attention si l'on veut réaliser des extensions. Par ailleurs ce régulateur n'est pas un modèle de précision. Il est fortement conseillé de vérifier la tension qu'il délivre sur les broches 33 et 34 du connecteur d'extension. Il faudrait très sérieusement s'inquiéter si cette tension était supérieure à 5,25 volts. Les RAM dynamiques 4164 sont

des composants extrêmement fragiles — une légère surtension suffit à les détruire — et ce sont également les composants les plus chers de l'ORMOS. Si donc la tension était trop forte, il faudrait soit consulter le revendeur, soit monter vous-même un autre régulateur, soit adapter une autre alimentation au système par les broches 33 et 34. Cela peut être plus approprié si vous désirez réaliser des extensions et évitera toute surchauffe due au régulateur à l'intérieur de l'appareil.

8 Le clavier de l'Oric

Nous ne le dénigrerons pas car nous avouons une certaine faiblesse pour lui. Il arrive cependant que certaines touches se coincent. On peut l'éviter en montant un film plastique sur la face avant de l'Oric empêchant d'appuyer trop à fond sur les touches, ou encore en fixant par-dessus la plaque portant les affectations des touches une autre plaque métallique ou cartonnée un peu épaisse adéquatement découpée. Il est ainsi possible de redéfinir rapidement tout le clavier (caractères graphiques, transformation AZERTY) alors qu'il faut coller des pastilles sur les touches des claviers d'autres types.

En cas de blocage fréquent, il faudrait ouvrir l'appareil et rectifier la position du clavier après en avoir desserré les vis de fixation.

9 Le VIA

C'est une des pannes graves les plus courantes de l'Oric. Attendu que les constructeurs semblent se montrer beaucoup plus confiants dans l'Atmos et que les deux appareils présentent exactement la même structure matérielle, nous pensons qu'il existe un défaut concernant la gestion du VIA dans la ROM 1-0. Ce VIA est si versatile qu'il peut se produire des situations conflictuelles sur le port A. Nous le répétons donc : pour garder le VIA viable, interdisez les interruptions chaque fois que c'est possible. Par ailleurs il ne faut pas poker n'importe quoi dans un circuit d'entrées-sorties, et ceci à plus forte raison si un périphérique quelconque est relié au système. En règle générale, il vaut mieux éviter de s'amuser avec les premières pages de la RAM et n'en modifier le contenu qu'à bon escient. Si un programme tournant sans les interruptions est susceptible d'opérer des modifications impromptues dans ces pages, il est conseillé de reprendre la main par un RESET programmé. *(V. 43, 44).*

10 Le magnétophone

L'ORMOS est incapable de sauvegarder un programme. Il ne sort qu'une modulation continue. Il peut par contre les charger. Cette panne assez rare est due à un mauvais positionnement de l'ULA sur son support. Il faut ouvrir l'appareil et enfoncer l'ULA à fond dans son support.

Enfin l'ORMOS dispose de deux vitesses de lecture-écriture. Si l'une est très rapide et d'une fiabilité souvent douteuse, l'autre, par contre, s'accommode des pires casseroles et est désespérément lente. S'il n'y a pas encore de Logo pour ORMOS, il y a au moins un mode tortue. Il ne faut pourtant par renoncer trop rapidement au mode FAST, ce n'est bien souvent qu'une affaire de réglage. Si vous êtes capable de lire une cassette enregistrée en mode FAST, vous devez pouvoir lire et écrire vos programmes sur cassette en ce mode sans problème.

Il faut d'abord mettre toutes les chances de son côté, nettoyer les têtes du magnétophone, les démagnétiser si possible, s'assurer s'il y a lieu que les piles sont neuves, n'utiliser que des cassettes de bonne qualité, vierges de préférence, en milieu de bande. Enfin il est bon de ne brancher pour ces essais que la fiche EAR en lecture, et que la fiche MICRO en écriture. Il vaut mieux ne pas utiliser la télécommande avec un gros magnétophone, un courant important risquant de griller le relais de l'ORMOS.

Voici la marche à suivre ensuite, si vous adoptiez notre méthode :

— se procurer une cassette convenablement enregistrée en mode FAST, un logiciel du commerce éventuellement, de préférence un programme Basic non protégé ;
— la lire à plusieurs reprises sur votre appareil en modifiant les réglages de volume de sortie et de tonalité s'il y a lieu. Sur un magnéto 6 volts, le volume de sortie doit avoisiner les 2/3. On peut insister sur les aigus ;
— noter les valeurs des réglages pour les lectures qui se sont déroulées sans faute et adopter des positions moyennes que l'on ne rectifiera en principe plus par la suite. Ne pas se fier à l'Errors found de l'Atmos, vérifier le programme, à l'exécution et/ou au listage ;
— si l'on n'arrive pas à un résultat correct, il faudra peut-être rectifier l'azimutage de la tête de lecture. C'est une opération un peu délicate, pas toujours évidente selon les appareils. On peut se faire aider par un spécialiste, en spécifiant bien l'usage de l'appareil car les exigences audio ne sont pas les mêmes. Sinon l'azi-

mutage se règle par une petite vis à côté de la tête, accessible par un trou ou une encoche. On peut faire comme précédemment plusieurs essais avec différents azimutages et déterminer une position optimale.

Admettons maintenant que nous ayons réussi à lire notre programme sans la moindre erreur de manière répétitive. Il s'agit maintenant de régler le magnétophone pour l'enregistrement, en se gardant bien de retoucher les réglages de sortie. Deux cas se présentent alors :

— le magnétophone dispose d'un contrôle manuel de niveau d'entrée, auquel cas on procédera à plusieurs sauvegardes d'un même programme avec différentes positions de ce réglage. On relira ensuite ces programmes, on notera le nombre d'erreurs pour chaque sauvegarde, et l'on choisira une position moyenne du réglage parmi celles pour lesquelles aucune erreur n'a été détectée ;
— le contrôle de niveau d'entrée est automatique. Si dans ce cas on a tout de même des problèmes de sauvegarde, c'est probablement parce que le niveau de sortie de l'ORMOS est trop fort. C'est moins grave que s'il était trop faible. On peut y remédier en montant un potentiomètre de 22 ou 47 kΩ (A) comme indiqué sur la figure 2. On procédera comme dans le cas précédent. On mesurera avec un contrôleur les valeurs des résistances du potentiomètre et l'on prendra les valeurs les plus approchées pour monter deux résistances en diviseur de tension comme indiqué sur

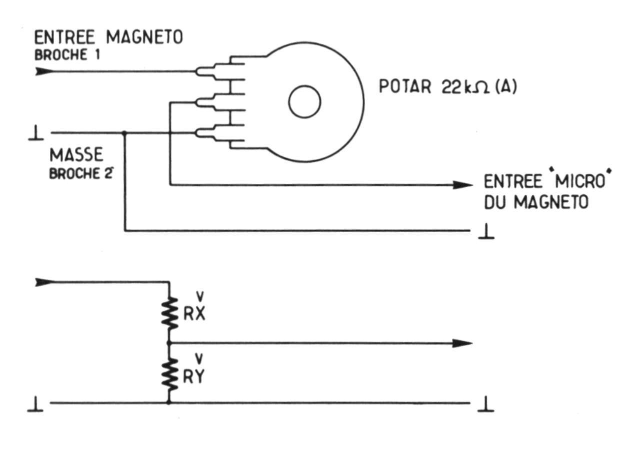
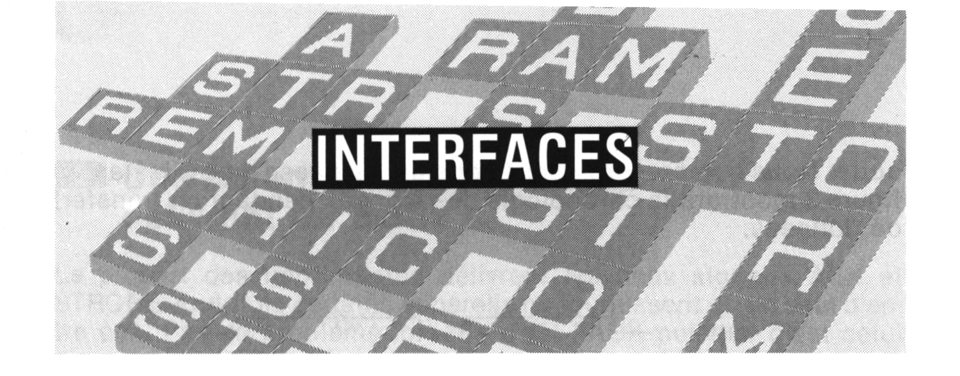


Fig. 2

le schéma. On arrivera à les caser sans trop de peine à l'intérieur de la prise DIN. Ces valeurs ne sont pas trop critiques, et si l'on veut s'éviter la peine de monter le potentiomètre, on peut tenter de monter directement un diviseur de tension avec deux résistances de 10 kΩ.

Toute cette procédure peut paraître un peu longue, mais il ne faut pas hésiter à la réaliser avec le plus grand soin ; cet effort sera largement rentabilisé par la suite. Les sauvegardes se font en effet huit fois plus vite qu'en mode lent. Il est souvent nécessaire de faire de nombreuses sauvegardes pendant la mise au point d'un programme ; on hésite pourtant quand on sait que cela va prendre plusieurs minutes, au risque de tout perdre par un plantage malencontreux.



Pour coupler un périphérique à un ordinateur il est nécessaire d'établir une voie de communication, en un mot il faut les interfacer. Il existe tellement de normes de bus et de protocoles de communication qu'il est difficile de s'y retrouver. En dehors des entrées-sorties indispensables à un système minimum (écran, clavier, cassette), il existe deux principaux standards de communication, l'interface parallèle Centronics et la liaison série RS-232.

11 Interface parallèle Centronics

C'est le mode standard le plus répandu pour connecter une imprimante. C'est le plus simple et le plus rapide.

Les 8 bits de données sont transmis en format parallèle sur 8 fils. 2 fils supplémentaires sont nécessaires pour les signaux de contrôle, l'un en entrée, l'autre en sortie. Un dernier fil doit assurer la masse. Dans les appareils sérieux toutes les lignes doivent en fait avoir leur retour de masse, c'est le cas de l'ORMOS. Des interférences sont en effet à craindre vu la fréquence des signaux. Nous avons donc déjà un minimum de 20 fils.

Il existe par ailleurs un certain nombre de signaux complémentaires reconnus par la norme Centronics, en entrée comme en sortie. Nous ne les décrirons pas en détail, l'interface parallèle de l'ORMOS ne pouvant pas les gérer. Leurs fonctions sont généralement accessibles également par logiciel.

Un de ces signaux nous intéresse néanmoins, le signal BUSY (occupé). Nous avons déjà vu *(V. 3)* le principe d'une communication parallèle en mode « Poignée de main ». L'imprimante possède en fait deux signaux distincts pour signifier à l'ordinateur qu'elle a digéré la dernière donnée et qu'elle est prête à en recevoir une

autre. Nous comprendrons mieux ce qui se passe en examinant la figure 3 montrant la chronologie des signaux pendant un transfert de données.

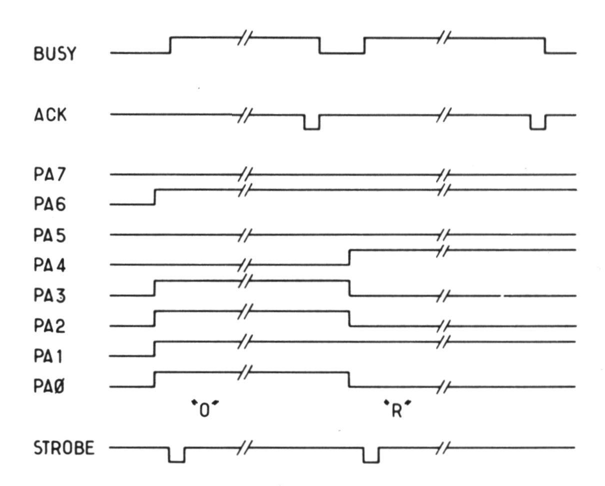


Fig. 3 - Transmission d'un « O » (01001111) et d'un « R » (01010010) sur le port parallèle.

La ligne BUSY est normalement au niveau logique bas, la ligne ACK (pour ACKnowledgement, accusé de réception) au niveau haut. Lorsque la donnée est prête et stabilisée sur le port parallèle, l'ordinateur envoie sur la ligne STROBE une impulsion négative. La sortie BUSY de l'imprimante va passer au niveau haut lors du flanc descendant du STROBE. L'imprimante va traiter la donnée, puis elle va signifier sa disponibilité en envoyant une impulsion négative sur ACK et en rétablissant la ligne BUSY au niveau haut.

Les autres signaux importants de la norme Centronics sont une entrée d'activation de l'imprimante, une entrée de réinitialisation, des sorties permettant de savoir si l'imprimante est activée, si elle a du papier, si elle a détecté une erreur...

Il faudrait un autre coupleur parallèle à l'ORMOS pour pouvoir gérer ces signaux *(V. 15)*, mais il est tout à fait possible de piloter une imprimante dans de bonnes conditions avec la seule interface parallèle de l'ORMOS.

12 Utilisation d'un périphérique délivrant un BUSY

La plupart des imprimantes délivrent les deux signaux ACK et STROBE, toutefois certains appareils ne fournissent qu'un seul d'entre eux. Pas de problème lorsque c'est l'ACK puisque c'est celui reconnu par l'ORMOS, mais cela veut-il dire qu'il faille renoncer à utiliser une imprimante ne délivrant que le seul BUSY ? Point du tout, la solution est très simple.

Nous avons vu que le VIA détectait une transition positive sur CA1 en mettant à 1 le bit 1 de son registre 13 *(V. 3)*. Mais il est pareillement capable de déceler une transition négative. Cette opération est commandée par le bit 0 du registre 12. Lorsqu'il est à 1 le VIA détecte un flanc positif sur CA1, et lorsqu'il est à 0 un flanc négatif.

Nous obtenons la valeur initialisée par le système dans ce registre en faisant ?PEEK(#30 C) et nous trouvons 221. Il suffit de faire POKE #30C,220 (ou POKE 780,220) pour être capable de détecter une transition négative sur CA1 et donc le rétablissement de la ligne BUSY au niveau bas. Il faut noter qu'un RESET va rétablir la valeur 221 dans le registre 12. On remarque aussi qu'en fait une impulsion ACK sera elle aussi détectée puisqu'elle possède un flanc descendant. On gagnera même quelques microsecondes. Nous n'avons observé aucune anomalie d'impression en faisant ce POKE. Le gain de temps, minime il est vrai, est cependant mesurable.

Application de ce fait : nous utilisons un petit plotter Canon X-710 qui présente exactement les mêmes caractéristiques que le MCP-40 associé normalement à l'ORMOS, même mécanique, même logiciel, mais pour un prix sensiblement plus attrayant, du moins pour l'acheteur. Hormis l'alimentation, la différence essentielle est que ce plotter délivre un BUSY. Nous obtenons d'excellents résultats avec lui. Il est toutefois ici absolument impératif d'inhiber les interruptions par un POKE 782,64 car il ne possède pas de buffer. Il est également possible d'utiliser la mini-imprimante thermique X-711 (toujours Canon et toujours pas chère). Nous donnerons en annexe le brochage du connecteur parallèle Canon. L'ORMOS pouvant éventuellement fonctionner sur piles, il devient possible d'emmener son micro favori aux champs...

13 Interface série RS-232 ou V-24

Il s'agit d'une liaison série, et si les parallèles se rencontrent souvent, les séries sont par contre rarement convergentes. Il existe une quantité impressionnante de normes de transmission série, et à l'intérieur de la seule norme RS-232 les options sont multiples.

Ne nous décourageons tout de même pas. Pourvu que l'on connaisse les exigences du périphérique et la manière par laquelle il signalera sa disponibilité, il sera possible de communiquer avec lui.

Dans une liaison série, les données sont transmises sur une seule ligne. Il existe un mode trois-fils, un pour la donnée émise, un pour la donnée reçue, et une masse commune. Les transmissions peuvent se faire simultanément (full duplex) ou alternativement (half duplex).

La liaison RS-232 est généralement asynchrone, c'est-à-dire que les données sérielles seront séparées par des intervalles variables, selon le temps de recherche et de préparation de ces données par l'émetteur d'une part, et le temps de traitement par le système récepteur d'autre part. Pour que ce système récepteur puisse s'y retrouver et différencier un niveau haut sur la ligne d'un bit d'information à 1, il est nécessaire d'envoyer au moins 2 bits supplémentaires, de niveaux opposés, 1 bit de départ pour signaler que le niveau présent sur la ligne au prochain top d'horloge sera le premier bit d'une donnée, et un bit de stop pour remettre la ligne à son niveau initial lorsque tous les bits d'une donnée auront été transmis.

Il faut se mettre d'accord avant de parler de niveau haut ou bas car les niveaux RS-232 sont inversés par rapport aux niveau TTL. La norme fixe les niveaux RS-232 :

— de —3 à —12 volts pour un niveau TTL haut,
— de +3 à +12 volts pour un niveau TTL bas.

Les connecteurs débiteront en principe des signaux à + et —12 volts, autorisant ainsi d'importantes pertes de niveau sur de longs câbles de transmission.

Essayons de nous y retrouver en examinant la figure 4 montrant ce qui se passe sur une ligne sérielle lors de la transmission d'un « O » et d'un « R ».

Toutes les opérations sont synchronisées avec la fréquence de la porteuse. Cette fréquence s'exprime en bauds, ou bits transmis par seconde. Si la transmission d'une donnée exige 10 bits, on ne pourra transmettre à 1 200 bauds par exemple qu'un maximum de 120 données par seconde.

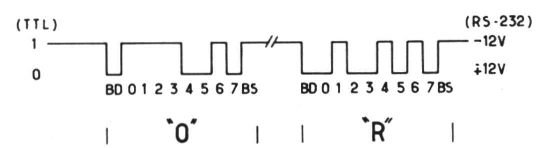


Fig. 4 - Transmission d'un « O » et d'un « R » sur une ligne série. Format : 1 bit de départ, 8 bits de données sans parité, 1 bit de stop.

A la norme TTL la ligne de transmission sérielle est au niveau 1 en l'absence d'émission. Le bit de départ BD (start bit) la mettra à 0, puis défileront les 8 bits de données par ordre croissant (DB0, DB1,...). Le bit de stop BS rétablira la ligne au niveau 1 en attente du prochain bit de départ.

Il va de soi qu'on ne peut expédier ainsi vers un périphérique des données en nombre illimité. Il viendra un moment où il ne sera plus capable de stocker les données, ou de les traiter. Il est donc nécessaire d'établir un protocole de communication.

Admettons que ce périphérique soit une imprimante. Plusieurs solutions couramment employées lui permettent de faire savoir si elle est prête ou non à recevoir une nouvelle donnée. La plus évidente est de répondre par sa propre ligne de transmission sérielle. On a essentiellement deux protocoles :

— **protocole XON-XOFF ou DC1-DC3 :** il faut d'abord souligner le fait que le buffer d'une imprimante RS-232 se vide en même temps qu'il se remplit, contrairement à ce qui se passe pour une imprimante parallèle où le logiciel attend généralement que le buffer soit plein pour optimiser les mouvements du chariot. Ceci parce que les données sont transmises bien plus lentement en mode série. Si la vitesse d'impression est inférieure à la vitesse de transmission des données, le buffer se remplira peu à peu. Avant que ça ne déborde, l'imprimante enverra vers l'ordinateur le code DC3 (CHR$(19)) lui signifiant d'arrêter la transmission. Lorsque son buffer sera presque vide, elle enverra le code DC1 (CHR$(17)) signifiant qu'elle est prête à recevoir de nouvelles données.

— **protocole ETX-ACK :** ici, on va envoyer vers l'imprimante un bloc de données de taille inférieure à la capacité du buffer de l'imprimante. On termine ce bloc par un code ETX (End Of Text ou CHR$(6)). L'imprimante va exploiter ces données puis expédier, lorsqu'elle rencontrera le code ETX, un code ACK ou CHR$(3),

signifiant qu'elle est prête à recevoir un nouveau bloc de données.

D'autres protocoles vont utiliser un ou plusieurs signaux supplémentaires, de rôle similaire au BUSY vu en mode parallèle. Deux protocoles encore :

— **protocole CTS :** Lorsque l'imprimante ne peut plus recevoir de données, elle met à 1 (niveau TTL) l'entrée CTS de l'émetteur, interdisant ainsi la transmission des données.

— **protocole DTR-DCD :** surnommé aussi Busy-Ready (nous l'appellerions plutôt Profanateur de sépulture). Il s'agit de mettre à 1 entrée DCD du sytème émetteur, ce qui peut déclencher une interruption.

Ce n'est pas tout d'avoir adopté un protocole de communication, il faut encore se mettre d'accord sur la nature des données à transmettre. Il y a encore ici une foultitude d'options, à savoir :

— le débit en bauds, bien sûr,
— le nombre de bits de données (de 5 à 8),
— la présence ou l'absence d'un bit de parité (paire ou impaire),
— le nombre de bits de stop.

Ne tentons pas d'approfondir ces points et posons tout de suite la question fondamentale : est-il possible de connecter l'ORMOS à un périphérique RS-232 ? La réponse est oui, pourvu que l'on sache sous quelle forme ce périphérique acceptera les informations qui lui seront transmises, et sous quel(s) protocole(s) il signalera son état.

Le VIA 6522 est tout à fait capable de communiquer en mode série, il ne s'en prive pas d'ailleurs puisque c'est lui qui assure l'interface magnétophone (synchrone). Il dispose de deux timers et d'un registre à décalage pour faciliter ces opérations. Il est donc très possible de l'utiliser pour une communication asynchrone, des exemples en sont donnés dans la littérature le concernant. Mais nous avons vu que le pauvre était déjà surchargé de travail. D'autre part on n'écrira généralement un programme (impérativement en langage machine) que pour une application donnée. Il peut être beaucoup plus intéressant de monter un ACIA 6551, circuit spécialisé dans les liaisons série, qui nous ouvrira la porte à toutes les possibilités de raccordement à toute la gamme des périphériques dotés de l'interface RS-232 *(V. 16)*. Ce composant est facile à configurer et programmer selon toutes les options vues plus haut. Les principaux signaux définis par la norme RS-232 sont présents sur ses broches. En voici la liste, avec leur rôle théorique et entre parenthèses le numéro de la broche correspondante sur le connecteur standard

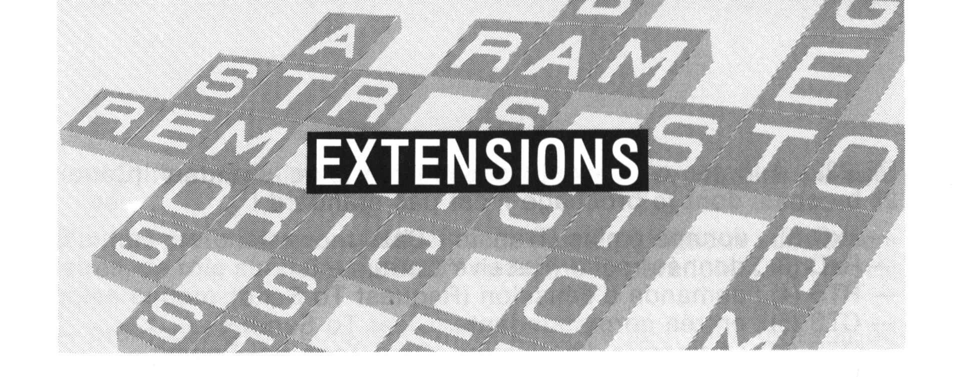
RS-232. A noter qu'il existe des connecteurs standard (Canon 25 broches) dont le brochage n'est pas standard !

— TxD (2) : donnée émise (Transmit Data),
— RxD (3) : donnée reçue (Receive Data),
— RTS (4) : demande d'émission (Request To Send),
— CTS (5) : entrée correspondante (Clear To Send),
— DTR (20) : demande d'activation (Data Terminal Ready),
— DSR (6) : entrée correspondante (Data Set Ready),
— DCD (8) : entrée recevant signal de détection de la porteuse (Data Carrier Detect),
— RxC (15 et 17) : c'est le signal d'horloge déterminant le débit en bauds. C'est une sortie ou une entrée, un ACIA peut se synchroniser directement sur ce signal.

Signalons aussi les brochages de la masse du châssis (1) et de la masse commune aux signaux (7).

A noter que ces signaux se présentent essentiellement par paires complémentaires, l'un en entrée, l'autre en sortie, vue la particularité de la liaison RS-232 autorisant émission et réception simultanées. Pour une communication bidirectionnelle entre 2 ACIA A et B on connecterait TxD(A) à RxD(B), RTS(A) à CTS(B), TxD(B) à RxD (A) et RTS(B) à CTS(A). Le rôle des autres signaux est avant tout une question de protocole.

Notons avant de quitter le sujet que si l'interface Centronics présente de grands avantages au niveau rapidité, on ne la rencontre que rarement sur les imprimantes de haut de gamme, et plus rarement encore sur les tables traçantes professionnelles. La raison en est que les périphériques deviennent de plus en plus performants et intelligents. Ils ont besoin d'envoyer à l'ordinateur d'autres signaux que le BUSY, éventuellement de lui transmettre également des données. Ce n'est pas possible par principe dans la norme Centronics. Ce n'est qu'un principe car les ports parallèles peuvent être programmés aussi bien en entrée qu'en sortie. Il suffirait d'un ou deux signaux supplémentaires pour que la communication puisse être bidirectionnelle. Il existe un autre type d'interface parallèle, l'IEE-488, qui peut transmettre les données dans les deux sens. D'autres normes de liaison série permettent d'accroître la rapidité de ce mode (RS-432 par exemple).



EXTENSIONS

Il ne faut pas s'effaroucher au départ : si l'électronique classique n'est pas si facile et si des connaissances relativement approfondies sont nécessaires dès qu'il s'agit de polariser un malheureux transistor, l'électronique digitale est paradoxalement bien plus simple. Les schémas internes des circuits intégrés (CI) peuvent comporter des centaines, voire des centaines de milliers de composants, mais ils sont conçus pour être d'une utilisation facile, particulièrement au sein d'une même famille. Le plus souvent un montage ne comportera que quelques CI, et peu ou pas de composants discrets.

Il y a tout de même quelques règles essentielles à respecter, d'abord pour ne pas endommager les CI :

— s'assurer qu'aucune broche des CI ne reçoit de tension indésirable, notamment lorsque le circuit nécessite plusieurs tensions d'alimentation ;
— ne jamais forcer à un niveau quelconque une sortie d'un circuit logique. Il arrive que ces sorties soient protégées contre de telles manœuvres mais ce n'est pas obligatoire ;
— être prudent dans l'emploi du fer à souder. Un fer à température régulée dont la panne sera reliée à la masse de l'appareil est conseillé. Il est absolument déconseillé par contre de toucher aux RAM 4164 qui sont des circuits très fragiles. Les autres composants de l'ORMOS sont assez costauds et nos bricolages sauvages et parfois hasardeux n'ont jamais réussi à les endommager. Toutefois, si vous n'êtes pas sûrs de vous, mieux vaut faire appel à quelqu'un sachant manier le fer. Une solution temporaire peut consister à monter le circuit sur plaque d'essai ;
— il sera prudent, lorsque c'est possible, de tester d'abord un montage à part et de vérifier qu'il se comporte conformément à ce que l'on en attendait avant de le connecter à l'ORMOS.

D'autres points sont à considérer si l'on veut voir le montage fonctionner :

— dans la plupart des extensions il faudra s'occuper des signaux PHI2 et R/W. Lorsque PHI2 est haut c'est le 6502 qui a accès au

bus, l'ULA lorsqu'il est bas. Par ailleurs cet accès se fait en lecture lorsque R/W est haut, en écriture lorsqu'il est bas. A noter que les circuits de la famille du 6502 posséderont généralement des entrées PHI2 et R/W ;
— la chronologie des signaux. Le cycle de lecture d'une RAM dure environ 500 ns. Le temps de propagation d'une porte TTL est de l'ordre de 30 ns. On ne peut pas multiplier indéfiniment les obstacles sur le chemin d'un signal ;
— les sortances des circuits à grande intégration sont faibles. On ne peut pas y connecter beaucoup d'entrées sans voir les tensions chuter. Il y a intérêt à utiliser des circuits LS ou mieux des CMOS lorsque c'est possible. Il peut être obligatoire d'intercaler des buffers sur des lignes très encombrées ;
— l'ORMOS consomme environ 700 mA. On peut en demander un peu plus au régulateur, mais il ne faut pas être trop exigeant. Il faudra avoir recours à une autre alimentation pour des montages gourmands. On peut songer, si l'on utilise des périphériques, à leur emprunter quelques mA ;
— des condensateurs de découplage peuvent être nécessaires lorsque des CI demandent des pointes de consommation élevées. Ces condensateurs doivent alors être soudés le plus près possible des broches d'alimentation des CI ;
— les signaux ont des fréquences élevées, les liaisons doivent être courtes, éventuellement blindées ;
— enfin, tous les signaux intéressants ne sont pas sur les connecteurs arrière, il sera dans certains cas obligatoire d'ouvrir l'appareil. Il pourra aussi être nécessaire d'interrompre des pistes. On peut, si l'on rechigne à employer le fer à l'intérieur de l'appareil, récupérer les signaux en enfichant des fils rigides de faible section dans les supports de l'ULA ou de la ROM(1). Pour interrompre une liaison avec une broche d'un de ces deux composants, on peut dégager le CI de son support et déplier délicatement la broche vers l'extérieur. Si l'on répugne à cette manœuvre, qu'il ne faut d'ailleurs pas répéter souvent au risque de rompre la broche, on peut intercaler un autre support entre le support original et le CI ; les opérations délicates seront alors effectuées sur ce support.

Les brochages des CI sont donnés en annexe.

---

(1) Certains ATMOS (venant peut-être d'une usine écossaise) n'ont pas de support pour l'ULA.

14 Décodage d'adresses

Le premier problème rencontré dans la réalisation d'une extension est de décider à quel emplacement mémoire on va la placer. Le problème est aux trois quarts résolu dans l'ORMOS par la page 3 réservée aux entrées-sorties. La broche 5 du connecteur d'extension passe au niveau bas lorsqu'une quelconque case de cette page 3 est adressée. Nous avons vu que dans la configuration initiale du système ce signal active le VIA qui n'occupe que 16 adresses. Il serait intéressant de pouvoir décoder les 4 adresses AB4 à AB7 pour délimiter 16 blocs de 16 adresses à l'intérieur de cette page.

Miracle ! Il existe un CI TTL qui peut réaliser tout seul cette opération, le 74(LS)154. Il possède 4 entrées binaires A, B, C et D — A correspondant au poids le plus faible — et 16 sorties numérotées de 0 à 15. Deux entrées E1 et E2 permettent en outre, lorsqu'elles sont toutes deux au niveau bas, d'activer le CI. Lorsqu'il est désactivé, toutes les sorties sont à l'état haut. Lorsqu'il est activé, la sortie correspondant à la valeur du quartet DCBA est basse, les autres sont hautes.

Il suffit donc de relier la broche 5 du connecteur à l'entrée A1, E2 étant relié à la masse, les signaux AB4 à AB7 (broches 20, 21, 23, 25) aux entrées A à D, et le tour est joué. Pas tout à fait cependant puisque le VIA occupe encore toute la page 3. Il s'agit de le désactiver pour les adresses supérieures à #30F. Lorsque la page 3 est sélectionnée, notre 74154 est activé. La sortie 0 est basse lorsque les bits d'adresse AB4 à AB7 sont tous à 0, elle est haute pour toutes les autres adresses. Ceci nous convient très bien, à un détail près : le signal d'activation du VIA présent sur le connecteur à la broche 6 obéit à la logique inverse. Un niveau haut activera le VIA, et vice versa. Il faudra donc intercaler entre la sortie 0 du 74154 et l'entrée CS1 du VIA une porte inverseuse de type 7404 par exemple. Voici donc un CI supplémentaire, mais signalons aux radins qu'une porte d'un 7404 à l'intérieur de l'ORMOS est inutilisée. Il y a toutefois mieux à faire : quitte à intervenir à l'intérieur de l'appareil on peut interrompre la liaison entre la broche 23 de l'ULA et l'entrée CS2 du VIA qui active le CI au niveau bas. On comprendra mieux les deux possibilités en examinant la figure 5.

Il y a bien sûr d'autres moyens de réaliser ce décodage selon les besoins. On peut ne décoder que 3 adresses et diviser la page 3 en 8 blocs de 32 adresses par exemple (on peut alors utiliser un 74(LS) 138 au lieu du 74(LS)154).

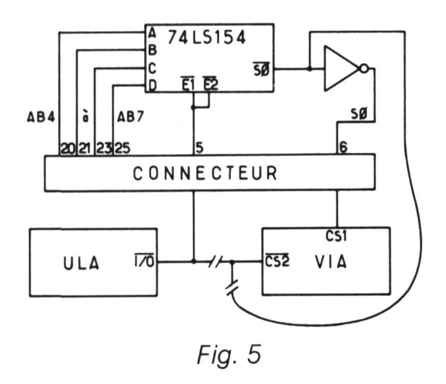


Fig. 5

Et si l'on ne désire monter qu'un seul CI d'entrées-sorties, seules deux portes NAND sont nécessaires *(V. 15).*

Les circuits logiques possèdent le plus souvent une entrée d'activation en logique négative ; pas de problème de ce côté donc.

Les circuits apparentés au 6502 possèdent des entrées PHI2 et R/W, mais si l'on emploie d'autres CI comme un simple latch pour se doter de quelques signaux de commande vers l'extérieur ? On peut relier l'entrée E2 au signal PHI1 (ou au PHI2 inversé) et établir une logique de commande du boîtier voulu tenant compte du signal R/W selon ce que l'on désire. Attention aux conflits sur le bus de données !

15 Une horloge parallèle (pour des réveils en série)

Le premier circuit d'entrées-sorties qu'il nous semble utile de monter est un autre coupleur parallèle, le VIA de l'ORMOS ayant déjà tant de travail qu'il est imprudent de le solliciter davantage. Nous disposerons ainsi de 16 lignes d'entrées-sorties supplémentaires. Nous pourrons en utiliser certaines pour gérer les signaux complémentaires de l'imprimante, par exemple, ou éventuellement la piloter entièrement à l'aide de ce coupleur et contourner ainsi les problèmes posés par l'interface parallèle de l'ORMOS.

Nous pourrions choisir de monter un autre VIA, bien sûr, mais il existe un CI plus récent de la même famille, le CIA 6526 (Complex Interface Adapter), qui possède d'intéressantes possibilités. Il serait

trop long de détailler ici toutes les fonctions de ce CIA et nous renvoyons pour cela à la brochure constructeur dont il est en principe possible d'obtenir au moins une photocopie à l'achat, ou aux ouvrages en traitant. Voyons cependant ce qui le distingue essentiellement du VIA :

— leurs brochages sont très voisins. On aura intérêt à monter le CIA sur un support dans l'optique de le remplacer facilement par un VIA pour certaines utilisations ;
— les ports A et B occupent respectivement les registres 0 et 1, plus logiquement que dans le VIA ;
— pas de signaux CA1, CA2, CB1, CB2 mais une sortie PC sur laquelle apparaît une impulsion négative lors d'une opération sur le seul port B. FLAG est une rentrée réagissant à un flanc négatif en positionnant le bit 4 du registre de contrôle d'interruptions (# D) à 1 ;
— le CIA possède un registre destiné aux communications sérielles (# C) auquel sont associées deux broches distinctes, CNT et SR ;
— outre deux timers 16 bits analogues à ceux du VIA, le CIA comporte une véritable horloge donnant les heures, les minutes, les secondes et même les dixièmes de seconde, le TOD (Time Of Day clock), sur lequel nous allons donner quelques précisions puisque c'est là la principale innovation par rapport au VIA.

Le TOD est complètement indépendant de l'horloge du système, car il comptabilise les impulsions sur la broche TOD, prévue pour recevoir un signal dérivé du secteur, à une fréquence de 50 ou 60 Hz (50 Hz = bit 7 du CRA (# E) à 1). Le TOD occupe quatre registres, à savoir :

— TOD/10 (8) : dixièmes de seconde,
— TOD SEC (9) : secondes,
— TOD MIN (10 ou # A) : minutes,
— TOD HRS (11 ou # B) : heures avec bit 7 = AM/PM.

Ces registres ont plusieurs fonctions, on peut y écrire soit pour une mise à l'heure, soit pour programmer une alarme (bit 7 du CRB (# F) à 1). Cette alarme peut générer une interruption au moment désiré. Une opération sur le registre 11 provoque le verrouillage des 4 registres TOD sur leurs valeurs présentes, mais le comptage continue parallèlement. Le redémarrage du compteur en lecture est assuré par une opération sur le registre 8.

Détail étrange : lorsque l'on consulte un registre du TOD, on peut s'étonner de trouver des valeurs supérieures à 59. Les anglo-saxons

se seraient-ils laissés emporter trop loin dans leur frénésie de décimalisation ? Non, les registres du TOD contiennent des valeurs BCD : les unités sont représentées par les 4 bits de poids faible, les dizaines par les bits 4 à 7. Pour obtenir une heure compréhensible, en prenant TD, TS, TM et TH représentant les adresses des registres /10, SEC, MIN et HRS, il faudrait faire quelque chose du genre :

```
H=(PEEK(TH)AND16)/1.6+(PEEK(TH)AND15)-12*(PEEK(TH)>127)
M=(PEEK(TM) AND 112)/1.6 + (PEEK(TM) AND 15)
S=(PEEK(TS) AND 112)/1.6 + (PEEK(TS) AND 15)
D=PEEK(TD)
```

sans oublier que la dernière opération est nécessaire même si l'on n'a pas besoin d'une telle précision. De telles opérations sont en fait plus simples en langage machine, le 6502 possédant un mode décimal plus approprié.

Passons au montage proprement dit, qui n'a rien de bien sorcier, le CIA n'ayant besoin d'autre composant extérieur qu'un éventuel condensateur de découplage. On connectera l'entrée CS à la sortie choisie du 74(LS)154. Si l'on choisit la sortie 1 les registres du VIA occuperont les adresses #310 à #31F. On connectera AB0 à AB3, DB0 à DB7, R/W et PHI2 aux signaux de même nom. On peut hésiter à connecter RES, car il est possible d'alimenter séparément le CIA, auquel cas l'horloge TOD continuerait à fonctionner quoi qu'il se passe du côté de l'ORMOS. Mais si RES est connecté, une remise en route de l'ORMOS ou un vrai RESET réinitialiseront tous les registres du CIA, y compris le TOD. Il faut se méfier aussi de l'IRQ, car l'ORMOS ne traite que les interruptions provenant du VIA. Si l'on établit la liaison il faudra réécrire une routine IRQ (ou une routine NMI car il est possible de relier la sortie d'interruption du CIA au NMI). Sinon les routines intéressant le CIA pourront lire directement le registre d'interruptions et attendre un FLAG par exemple.

Il faut encore, si l'on veut utiliser le TOD, lui fournir un signal à 50 Hz dérivé du secteur pour profiter de la précision journalière de celui-ci. Une solution simple et originale consiste à se servir de la haute impédance d'entrée des circuits CMOS. Une entrée d'un tel CI laissée libre oscille à la fréquence du champ ambiant, c'est-à-dire 50 Hz dans la plupart des environnements où nous utiliserons notre ORMOS. Des fréquences indésirables peuvent toutefois se superposer au 50 Hz, on les éliminera en les dérivant vers la masse par un condensateur d'assez forte valeur. Notons que si l'on utilise un CMOS à portes inverseuses, l'une de ces portes peut nous fournir l'inverseur dont nous avions besoin entre la sortie 9 du 74154 et l'entrée CS1 du VIA.

Il y a encore mieux à faire : si l'on ne veut monter qu'un seul CI d'entrées-sorties, il est possible d'effectuer le décodage d'adresses avec uniquement 2 portes NAND. On peut donc monter un CIA opérationnel avec seulement un 4011 B à 4 portes NAND, un condensateur de un microfarad pour filtrer le 50 Hz, et un éventuel condensateur de découplage.

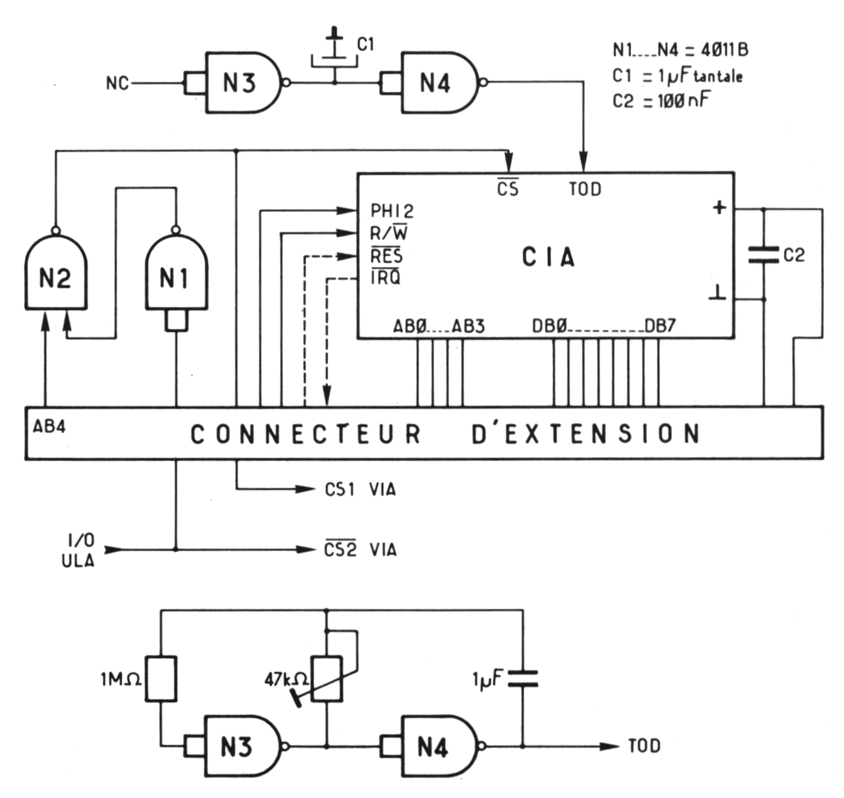


Fig. 6 - Montage du CIA avec un seul CI supplémentaire. Alternative de montage des portes N3 et N4 du 4011 B.

Cette possibilité est illustrée sur la figure 6. A noter qu'il n'est en principe pas du tout recommandé de laisser une entrée d'un CMOS « en l'air ». Il y a un risque de détérioration du composant dû à l'électricité statique, mais si ce risque était loin d'être négligeable avec

la série A, les CMOS série B sont bien plus résistants ; de fait nous n'avons jamais réussi à en griller un malgré les traitements que nous leur infligeons. Le risque existe néanmoins et l'on peut l'éviter en montant ces deux portes NAND en oscillateur plus classique, de période T = 0.7 R/C. On peut choisir une fréquence de 60 Hz qui évitera une opération logicielle.

Sinon, si l'on veut profiter de la précision du secteur (les quartz à 50 Hz sont rares), il faudra prendre garde à n'amener que des niveaux TTL sur la broche TOD. Un coupleur optique est recommandé.

16 Montage d'un CI d'interface série

L'ACIA 6551 (Asynchronous Communication Interface Adapter) nous ouvre l'accès à tous les périphériques RS-232. Une fois de plus nous n'allons pas recopier la brochure du constructeur, et nous borner à l'essentiel.

Le montage de l'ACIA ne pose guère plus de problèmes que celui du CIA. On connectera l'entrée d'activation CS1 à la sortie voulue du 74(LS)154, à moins que l'on ne réalise le décodage d'adresses au moyen de 2 portes NAND, comme dans la section précédente. L'ACIA possède une autre entrée d'activation, CS0, qu'il faut porter au niveau haut.

On connectera AB0, AB1, DB0 à DB7 ; RES, PHI2 et R/W broches correspondantes du connecteur de l'ORMOS. On adoptera les mêmes restrictions sur l'IRQ que dans la section précédente. Il n'y a que 2 entrées d'adresses car l'ACIA ne possède que 4 registres.

Enfin un composant extérieur est nécessaire, un quartz standard de 1, 8432 MHz à monter entre les broches X1 et X2. Le sempiternel condo de découplage, de 100nF par exemple, est en outre conseillé, à monter entre les broches 1 et 15.

La liaison établie du côté de l'ORMOS, il reste à établir la connexion avec le périphérique ; c'est un peu moins simple et cela va dépendre des cas.

Le premier problème est une question de niveau. L'ACIA ne fournit et n'accepte que des niveaux TTL alors qu'il y a toutes chances pour que les signaux présents sur le connecteur du périphérique soient au standard RS-232 (+ et —12 volts). Si l'on veut donc disposer d'une véritable interface directement utilisable avec de nombreux périphériques, il va falloir faire les conversions. Il existe heureusement des CI spécialisés.

Le MC 1488 (ou pas mal de choses finissant par 88) comporte 4 buffers-inverseurs transformant les niveaux TTL en niveaux RS-232. Il faut l'alimenter avec du + et —12 volts. Les tampons ont une ou deux entrées absolument équivalentes, on peut n'en connecter qu'une et laisser l'autre en l'air.

Le MC 1489 (ou pas mal de trucs terminés par 89) commet l'opération inverse. Il possède des entrées RC (Response Control) que l'on peut s'abstenir de connecter. Elles servent à faire varier dans une faible mesure le seuil de déclenchement des portes inverseuses.

Il faut donc relier les sorties du 1489 aux entrées de l'ACIA et les sorties de l'ACIA aux entrées du 1489 qu'il ne faut pas oublier d'alimenter (2 piles de 9 volts peuvent suffire pour une utilisation ponctuelle). Nous verrons plus loin quels signaux nous allons effectivement exploiter.

Mais il n'est pas impossible que le périphérique auquel on envisage de se connecter soit au niveau TTL (c'est au moins le cas de quelques micros tels le VIC, le C64 ou le Canon X-07). Pourquoi effectuer alors deux conversions coûteuses puisque la liaison TTL directe marche admirablement. De même, si l'on ne monte l'interface que pour utiliser un seul périphérique, il n'est pas interdit d'aller soulever le capot de celui-ci à la recherche d'un couple probable de 88-89. Si d'aventure la conversion était plus discrète, on ne manquerait pas d'identifier le circuit d'interface utilisé, un ACIA, SIO ou autre UART de brochage connu. Il ne resterait alors qu'à connecter directement les entrées et sorties correspondantes des circuits d'interface.

Il reste à savoir quels signaux nous allons exploiter, que ce soit en TTL ou RS-232. C'est avant tout une question de protocole. Il faudra établir dans tous les cas les 2 liaisons TxD et RxD (mode minimum 3 fils). Pour le reste la présence effective des signaux sur le connecteur n'est pas toujours nécessaire ni même parfois souhaitable. S'il s'agit d'une imprimante gérant le protocole CTS il faudra relier le CTS de notre ACIA à la ligne correspondante. Si c'est le protocole DTR-DCD il faudra connecter le DCD... Si nous n'avons pas relié l'IRQ nous pouvons ne pas nous préoccuper des entrées DCD et DSR, il n'en va peut-être pas de même du côté de l'imprimante. Il faut configurer l'ACIA selon les exigences du périphérique. Il faudra peut-être procéder par tâtonnements avant d'arriver à une première impression (méfiez-vous, ce n'est pas toujours la bonne). Attention surtout à ne pas balancer des niveaux RS-232 sur des broches TTL !

Imaginons que nous ayons franchi cette autre étape, il va maintenant falloir mettre au point un programme de transmission. C'est

encore avant tout une question de protocole. Nous invitons le lecteur à consulter l'ouvrage *Programmation en assembleur 6809* par Bui Minh Duc, qui décrit en détail la programmation en langage machine de l'interface RS-232 sous de multiples protocoles. Il s'agit malheureusement d'un autre ACIA et d'assembleur 6809, mais il sera très possible d'adapter les programmes proposés.

Voici tout de même un exemple d'application à peu près universel, pour montrer qu'au bout du compte la tâche n'est pas aussi compliquée qu'on pourrait l'imaginer. Le truc consiste à ne pas se soucier du protocole. On établit alors uniquement la liaison TxD. On met les entrées CTS, DSR et DCD de l'ACIA au niveau bas. On s'assure qu'il en va de même pour l'imprimante. Admettons que nous ayons commandé notre ACIA par la sortie 2 du 74154, ses registres occuperont alors les adresses 800 à 803 (#320 à #323). Il faudra programmer les registres 2 et 3 selon le débit et le format exigés par l'imprimante. Puis on stockera les données à expédier en page 4 par exemple, de #400 à #4FF ; nous supposons ici que le buffer de l'imprimante a une capacité d'au moins 256 octets. On peut programmer la transmission de ce bloc ainsi :

```
1000 FOR A=1024 TO 1279
1010 POKE 800, PEEK(A)
1020 IF (PEEK(801) AND 16) = 0 THEN 1020
1030 NEXT
```

On préparera ensuite un nouveau bloc et l'on introduira éventuellement une temporisation pour permettre à l'imprimante d'en finir avec le dernier bloc de données. La ligne 1020 teste le bit 4 du registre d'état pour savoir si l'émission de la donnée précédente est terminée.

Ce n'est pas plus difficile. Dans certains cas la préparation des données sera longue par rapport aux vitesses de transmission et d'impression, il sera alors possible de n'avoir rien d'autre à faire que poker la donnée en 800.

La programmation en mode bidirectionnel sera certes plus ardue, mais ce n'est nullement une tâche insurmontable. On fera généralement une bonne affaire en évitant d'acheter une carte d'interface (souvent fort cher si elle existe) ne contenant qu'un ACIA ou autre UART, les CI 88 et 89, et éventuellement un peu de logiciel.

Toutefois il peut toujours y avoir des surprises. Nous ne conseillons pas d'acheter un périphérique RS-232 avant d'avoir vérifié que l'on est bien capable de communiquer selon cette norme. Il faut aussi s'assurer que la documentation du périphérique spécifie sans

trop d'équivoque toutes les modalités de la communication, protocoles, débit, format, parité, brochage du connecteur, taille du buffer.

17 Montage du KGB, ou l'ORMOS 64 K

Le KGB, cela signifie Kolossal Gaspillage de Bits, notre montage vise en fait à y remédier.

Nous savons donc d'une part que la RAM de l'ORMOS est contenue dans 8 boîtiers 4164, ce qui représente 64 K, d'autre part que nous n'avons accès qu'à 48 K. Que font les 16 K restant occultés par la ROM ? Rien. Existe-t-il un moyen quelconque d'y accéder par le système ? Peu d'informations ont été fournies sur ce sujet. Il semble que ce soit impossible par seul logiciel ; on sait uniquement que l'on peut en portant l'entrée MAP à 0 désactiver la ROM et activer les 16 K de RAM correspondants. Nous y reviendrons dans la prochaine section, mais cette méthode impose au système d'être contrôlé par un programme en langage machine extérieur à cette zone. Par ailleurs l'affichage est extrêmement perturbé. Notre montage permet d'accéder à ces 16 K, en lecture comme en écriture, à partir du Basic, en gardant la ROM, l'affichage et toutes les fonctions de l'ORMOS.

Au niveau de la RAM, il faut et il suffit pour lire ou écrire dans le bloc qui nous intéresse de lui fournir une adresse comprise entre # C000 et # FFFF, c'est-à-dire une adresse dont les 2 bits de poids fort seront à 1. Nous avons vu que lorsque l'ULA décodait une adresse de ce type, elle ne fournissait pas les impulsions de validation RAS et CAS et activait la ROM. Notre truc va consister à tromper l'ULA en lui fournissant une adresse pour laquelle elle validera la RAM, mais en forçant les bits UB14 et UB15 à 1 au niveau du multiplexeur commandant la RAM.

En examinant la carte de la mémoire de l'ORMOS, nous constatons que les 2 bits d'adresses de poids fort AB15 et AB14 divisent la mémoire en 4 blocs de 16 K, le bloc 0 de # 0000 à # 3FFF (0-0), le bloc 1 de # 4000 à # 7FFF (0-1), le bloc 2 de # 8000 à # BFFF (1-0), et le bloc 3 de # C000 à # FFFF (1-1) correspondant à la ROM. Nous ne pouvons pas toucher au bloc 0 car il contient toutes les variables-système, la pile du 6502, et éventuellement le programme Basic. Il n'est guère intéressant de toucher au bloc 2 qui contient l'écran et les tables de caractères. Nous avons donc choisi le bloc 1, et notre montage va permettre d'accéder au bloc de RAM 3 en adressant le bloc 1.

Pour que cette opération présente une utilité notable, il est nécessaire qu'elle soit commutable par logiciel, c'est-à-dire que l'on puisse accéder aussi normalement au bloc 1, de manière à disposer effectivement des 64 K désirés. Il nous faudra donc un signal de commande. Pas de problème si vous avez déjà monté un second coupleur parallèle, le CIA pourra collaborer avec le KGB et lui fournir un bit de commande, mais nous avons voulu réaliser un montage indépendant pouvant de plus se loger à l'intérieur de l'ORMOS. Nous nous sommes donc intéressés de très près au PB5 du VIA.

Nous avons déjà signalé qu'il était inutilisé par le système ; il vient donc à l'esprit de s'en servir comme signal de commande. Profitons-en puisque de toute manière nous sommes obligés d'ouvrir l'ORMOS pour interrompre la liaison UB15. Un problème se pose cependant : si ce bit est inutilisé, il est néanmoins géré par le système qui le remet à 1 à chaque interruption. Nous intercalerons donc sur le chemin de notre signal de commande une bascule qui sera sensible à une transition négative sur PB5. Nous en avons dit suffisamment pour pouvoir présenter le schéma du KGB *(figure 7).*

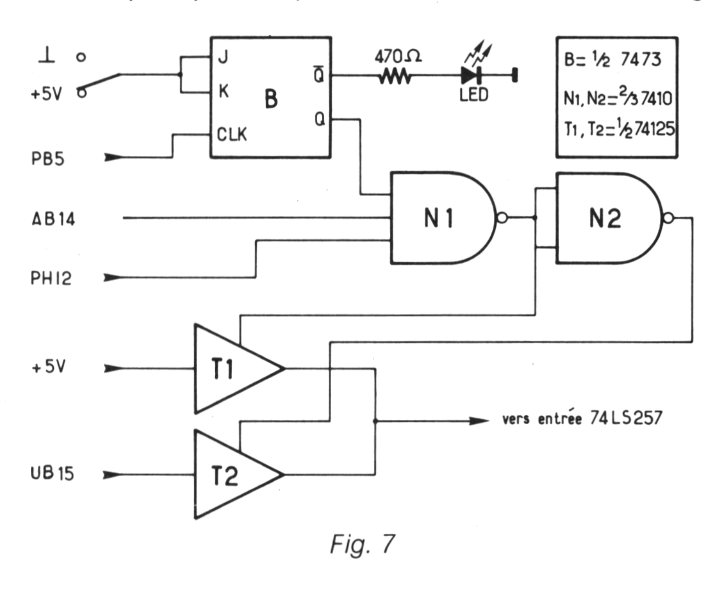


Fig. 7

Nous avons donc interrompu la liaison entre la broche 39 de l'ULA et la broche 14 du 74257 situé à côté du 7404. Cette interruption peut se faire en dégageant l'ULA de son support. L'entrée du 74257 reçoit

les sorties de deux buffers trois états T1 et T2. Une seule de ces sorties sera active à la fois car les entrées de commande de T1 et T2 sont inversées l'une par rapport à l'autre par la porte NAND N2. Il faudra satisfaire à trois conditions pour activer T1 par un niveau logique 0 en sortie de la porte N1 à 3 entrées. Il faudra d'abord que notre signal de commande soit haut, de même que AB14. C'est en effet ce bit qui indique qu'une case du bloc 1 est adressée. Pour des raisons de chronologie des signaux il faut prendre AB14 plutôt que UB14. A noter que ce bit est aussi à 1 lors de l'adressage de la ROM. C'est sans importance puisque l'ULA n'active pas la RAM pour ces adresses. Enfin il faudra que PHI2 soit haut, sinon on aurait des perturbations à l'affichage car lorsque PHI2 est bas c'est l'ULA qui lit la RAM.

Le signal de commande sera donc fourni par la sortie Q d'une bascule JK qui changera d'état à chaque transition négative sur son entrée d'horloge reliée à PB5 (broche 15 du VIA) lorsque ses entrées J et K seront à 1. Détail important : cette bascule doit être un TTL 7473 et non un 74 LS 73 qui ne possède pas la même table de vérité. Les circuits TTL « normaux » et LS sont souvent interchangeables, mais ce n'est pas une règle absolue. On peut par contre utiliser un 74 LS 10 et un 74 LS 125, c'est même préférable car ils absorberont moins de courant que leurs confrères TTL.

En option, une LED reliée à l'autre sortie de la bascule peut indiquer à l'utilisateur quel bloc de RAM est sélectionné (1 ou 3). On peut également monter un inverseur permettant de porter les entrées J et K au niveau bas. Ceci autorise le verrouillage du système dans l'une ou l'autre configuration. Attention, ce blocage ne portera pas sur l'état du système à cet instant, mais sur l'autre. C'est-à-dire que si l'on est sur le bloc 3 et que l'on porte les entrées J et K à la masse, le système sera bloqué sur le bloc 1 après une nouvelle impulsion sur PB5. D'autres options sont possibles : étudier pour cela le fonctionnement du 7473.

On peut aisément loger les trois CI sur une plaquette pouvant trouver sa place à l'intérieur de l'ORMOS, à côté de la ROM.

Passons maintenant à l'utilisation du KGB. Du fait que le registre du port B du VIA est automatiquement remis à jour à chaque interruption, on peut commander le basculement du 7473 par un simple POKE #300,0 (ou POKE 768,0). Si l'on fonctionne avec les interruptions inhibées, il faudra faire

```
POKE 768,PEEK(768) AND 223 : POKE 768, PEEK(768) OR 32
```

ou l'équivalent en langage machine.

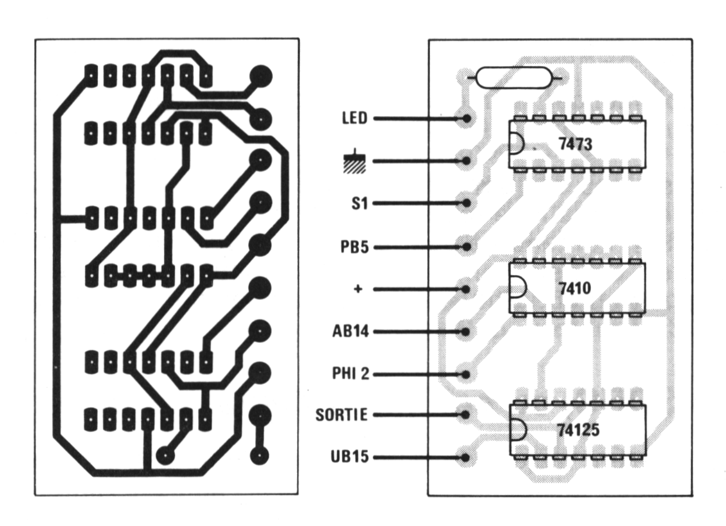


Fig. 8 - Suggestion de circuit imprimé et implantation des composants du KGB.

Un petit détail cependant : les instructions CSAVE et CLOAD font aussi basculer notre montage. Il suffit de le savoir pour trouver des parades ; notre inverseur va nous permettre par exemple de bloquer le système dans la configuration voulue avant une telle manœuvre. Cet inverseur est pratiquement nécessaire si l'on veut charger des programmes protégés qui se chargent en plusieurs fois. Il faut songer aussi, si l'on veut se servir du montage avec un utilitaire tel le moniteur 1.0, que celui-ci débute en # 7800, c'est-à-dire dans la zone litigieuse. Il faudra le charger deux fois.

Nous n'allons pas débattre de l'utilité du KGB, ceux qui se sentent à l'aise dans leurs 48 K n'auront certes aucun besoin de le monter. En bref 48 K c'est bien mais 64 K c'est mieux. Voici cependant quelques suggestions et remarques :

— on peut stocker plusieurs écrans TEXT ou HIRES dans ces 16 K pour agrémenter des jeux graphiques ;
— les graphismes sur imprimante demandent énormément de place en RAM et 16 K supplémentaires ne seront pas superflus ;
— le plus souvent, un programme Basic sera limité à 16 K par un HIMEM # 4000. Mais nous verrons qu'une bonne connaissance du fonctionnement des pointeurs peut permettre au Basic de sauter la zone litigieuse et/ou d'y stocker deux jeux de variables distincts *(V. 26,30)*. On peut par exemple faire

```
DOKE#9C,#4000:CLEAR:DIM A(1000):POKE 768,0
```

et recommencer la même opération pour disposer de deux tableaux identiques. On passera de l'un à l'autre par un POKE 768,0 ;

— l'un des blocs de 16 K est totalement à l'abri du Basic, notamment de la drastique routine d'initialisation qui remplit la RAM de 85. Son contenu ne peut être effacé que par l'extinction de l'appareil. On peut donc avoir un programme implantable en le chargeant ou en le transférant dans le bloc inaccessible.

18 Le SDEC, ou comment déplanter la bécane

Un inconvénient majeur de l'ORMOS est de ne pas disposer d'un RESET « tiède » qui réinitialiserait toutes les adresses stratégiques en pages 0 à 3 sans toucher au contenu de la RAM utilisateur. Le SDEC (pour Si Dallas m'Etait Conté : comprenne qui peut) permet de reprendre la main en toutes circonstances, sans exception, et de faire bien d'autres choses encore. Si vous ne vous sentez pas intéressé, notez avant de quitter cette section que le SDEC peut très bien être monté puis connecté à un ORMOS planté, et qu'il serait donc capable de récupérer ce programme sur lequel vous auriez trimé pendant des heures jusqu'à ce que survînt cet inexplicable et inextricable plantage...

Le SDEC a en fait plusieurs versions, correspondant à des exigences différentes. Lorsqu'un RESET chaud (NMI) est inefficace, il peut y avoir essentiellement trois causes :

— les points d'entrée aux routines d'interruption en RAM ont été modifiés,
— certaines adresses stratégiques en page 0 ont été modifiées,
— le 6502 est bloqué sur un octet qu'il ne peut interpréter. Les interruptions NMI et IRQ sont alors inopérantes.

Dans ces trois cas, la seule manœuvre possible est un vrai RESET équivalent à éteindre et rallumer l'ORMOS. Pas tout à fait cependant, car l'une des premières opérations de cette routine est de restaurer les points d'entrée en RAM des routines d'interruption (et autres sur ATMOS). On peut donc déclencher un RESET dès le saut NMI restauré, et le SDEC 000 consiste précisément à ne rien monter. Il s'agit seulement de mettre la broche 4 du connecteur d'extension à la masse et d'appuyer sur le RESET quelques millisecondes plus tard. Essayez donc...

En montant deux poussoirs RES et NMI (vrai et faux RESET), on peut avec de l'entraînement arriver à ne perdre que quelques centaines d'octets, voire moins.

On peut réaliser électroniquement cette temporisation. Il faut savoir que l'entrée RES est activée par un flanc ascendant, tandis que NMI est activée par un flanc descendant, et que RES est prioritaire sur NMI. Nous n'insisterons pas sur ce montage qui n'est toujours pas efficace à 100 %.

Nous donnons à titre d'exemple, en figure 9, le SDEC 004 qui substitue au niveau de la ROM le vecteur NMI au vecteur RES :

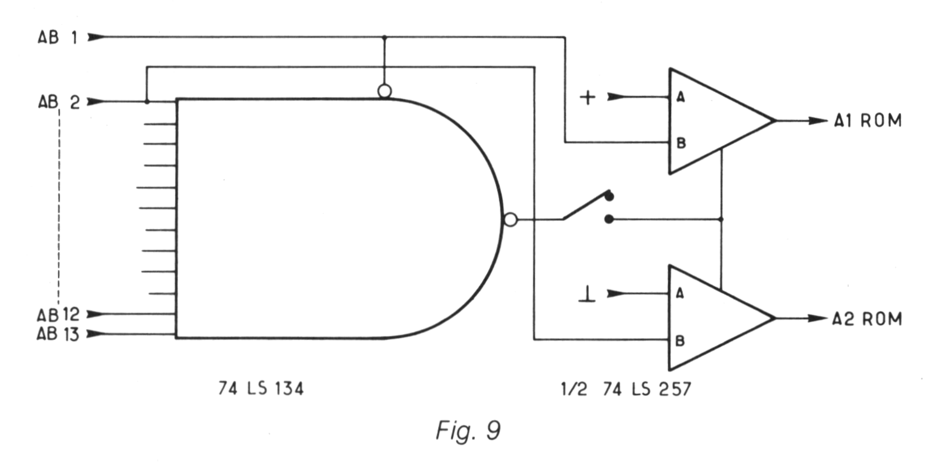


Fig. 9

Ce montage est bien sûr totalement inefficace si la routine NMI a été détournée. Nous ne le présentons que comme démonstration de ce que l'on peut faire avec deux CI.

Nos montages suivants ont visé à décoder les adresses # FFFC-# FFFD sur le bus, à inhiber pour ces adresses la ROM et à envoyer sur le bus de données une adresse intéressante, l'adresse de départ du NMI en ROM par exemple.

Le SDEC 007 peut décoder n'importe quel couple d'adresses en ROM (pourvu qu'il soit pair-impair), et substituer aux valeurs en ROM d'autres valeurs quelconques. Le montage se décompose en deux blocs bien distincts *(figure 10)*.

Les 4 CI 74 LS 266 décodent les 15 adresses AB1 et AB15 ainsi que le signal R/W. Chaque signal est appliqué à l'une des entrées d'une porte NOR exclusif, l'autre entrée servant de preset. Lorsqu'il y a

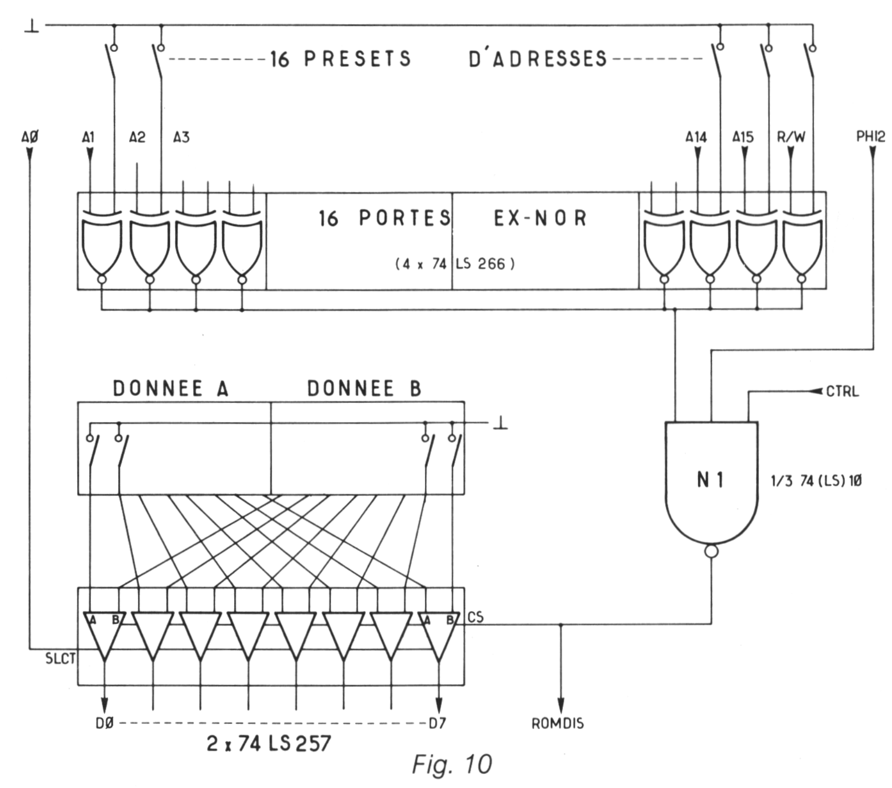


Fig. 10

concordance la sortie est à 1. Les seize sorties sont reliées ensemble, car les 74 LS 266 sont des CI à sortie à collecteur ouvert. On peut relier ensemble à peu près autant que l'on veut de ces sorties, un seul 0 en sortie d'une porte imposant un 0 en sortie générale. Cette sortie ne sera donc à 1 que lorsque les seize conditions imposées par les presets seront remplies. Cette sortie est récupérée par une porte NAND à trois entrées, qui reçoit aussi le PHI2 et un éventuel signal de commande externe.

Lorsque les seize conditions sont remplies, que PHI2 et le signal de commande sont hauts, la sortie du NAND passe à 0. Elle est appliquée à l'entrée ROMDIS qui désactive la ROM, et aux entrées d'activation de deux quadruples multiplexeurs 74 LS 257. C'est l'adresse A0 qui va valider en sortie l'une ou l'autre des entrées de chaque multiplexeur. Les 8 sorties sont reliées au bus de données. Les entrées A représenteront l'adresse de poids faible, les entrées B celle de poids fort.

Nous n'insisterons pas sur la réalisation pratique, car nous allons voir que ce montage est encore loin d'être définitif. Pour une manipulation aisée des presets, il serait judicieux de monter quatre octuples contacteurs DIL, deux pour les adresses et le R/W, un pour la donnée A, un pour la donnée B. Rappelons qu'une entrée TTL en l'air est au niveau haut, ce qui évite de câbler trente-deux résistances de rappel. Ici le preset de R/W doit être toujours haut. Pratiquement, si l'on n'envisage de décoder que le couple # FFFC- # FFFD, il suffit de mettre le preset de A1 à la masse.

Pour un déplantage absolu, il faudrait réinitialiser la machine en évitant d'appeler la routine de vérification de la RAM qui la remplit de 85. Cette routine a pour adresse # FA06 sur Oric 1 et # FA14 sur Atmos. Il suffit donc d'établir les presets d'adresses pour # FA06 ou # FA14 et de préparer un RTS (# 60) en donnée A, B étant ici indifférent. On ne peut pas ainsi récupérer directement le programme, car les pointeurs ont été réinitialisés comme par un NEW, mais ce n'est pas un problème *(V. 27).*

Les applications sont multiples ; nous n'allons pas en donner d'autres, car il y a encore mieux à faire.
Ce montage était conçu pour secourir un appareil planté aussi bien qu'un programmeur au bord du suicide. Le SDEC 007 peut en fait fonctionner avec n'importe quel système à base de 6502, et éventuellement à base d'autres CPU (6809 par exemple), mais l'ORMOS présente une intéressante particularité : les 16 K de RAM occultés par la ROM.

L'entrée MAP (Memory Access Priority) inverse en quelque sorte la configuration de l'ORMOS lorsqu'elle est activée par un niveau bas. Pour les adresses correspondant à la ROM, celle-ci est désactivée, et l'on a alors accès au bloc 3 de RAM, tandis que pour les autres adresses c'est la RAM qui est désactivée, le bus de données étant alors disponible pour de la mémoire externe.

Nous pouvons très bien utiliser notre montage avec le MAP au lieu du ROMDIS, à quelques détails près :
— il semble que MAP ne soit actif que s'il est appliqué pendant les deux phases d'horloge. Il ne faut donc plus appliquer PHI2 à l'entrée de la porte NAND,
— il faut bien sûr ne pas monter les 74 LS 257, ou les désactiver, pour éviter le conflit sur le bus de données avec le bloc 3 de RAM activé par MAP,
— le montage peut être actif en lecture comme en écriture,
— on peut ici décoder une adresse en RAM normale, et lui substituer en lecture les presets de nos 74 LS 257,

— on peut décoder des blocs de mémoire plus importants, il suffit pour cela de ne pas relier un signal d'adresse à l'entrée d'une porte EX-NOR. On peut avoir aussi le montage actif à la fois en lecture et en écriture pour ces adresses en ne connectant pas R/W.

Les possibilités sont encore ici énormes. Les applications vues précédemment sont possibles pourvu que l'on prenne la peine de décoder les adresses mentionnées en lecture écriture et d'y poker préventivement le RTS. On peut aussi interdire l'écriture sur l'adresse # 30 *(V. 45)* pour utiliser librement LPRINT, ou sur les sauts NMI et IRQ.

Les possibilités matérielles sont aussi immenses, on pourrait par exemple activer un VIA ou une RAM statique au lieu des 74 LS 257.

19 Buffer or not buffer

L'imprimante est un périphérique lent par rapport au micro. Un buffer permet de stocker sans perte de temps toutes les données envoyées sur le port parallèle, puis de les restituer en libérant le micro pour d'autres activités. Nous avons imaginé un buffer d'une capacité de 2 ou 4 K qui nous a paru séduisant par sa simplicité *(figure 11)*.

Décrivons rapidement le fonctionnement de ce buffer entièrement réalisé en technologie CMOS.

En écriture on met le compteur 4040 à 0, puis on envoie les données ; l'impulsion de retour ACK est donnée directement par le STROBE qui avance également le 4040 d'une unité, sélectionnant ainsi l'adresse suivante.

En lecture, on ferme la communication avec le micro, on ouvre celle avec l'imprimante, on met les RAM en lecture et on commute l'entrée d'horloge du 4040 (après l'avoir remis à 0) sur un oscillateur à base de portes NOR. Chaque impulsion va avancer le compteur d'un cran, puis envoyer le STROBE vers l'imprimante. Le BUSY va bloquer l'oscillateur. La fréquence de cet oscillateur est à calculer en fonction de la fréquence maximale de réception des données de l'imprimante. Cette fréquence est généralement relativement faible par rapport aux possibilités de transmission parallèle, de l'ordre du kilohertz.

Deux remarques : la première donnée, à l'adresse 0, risque de ne pas être transmise. Il faudra donc débuter par un NUL. Enfin, rien

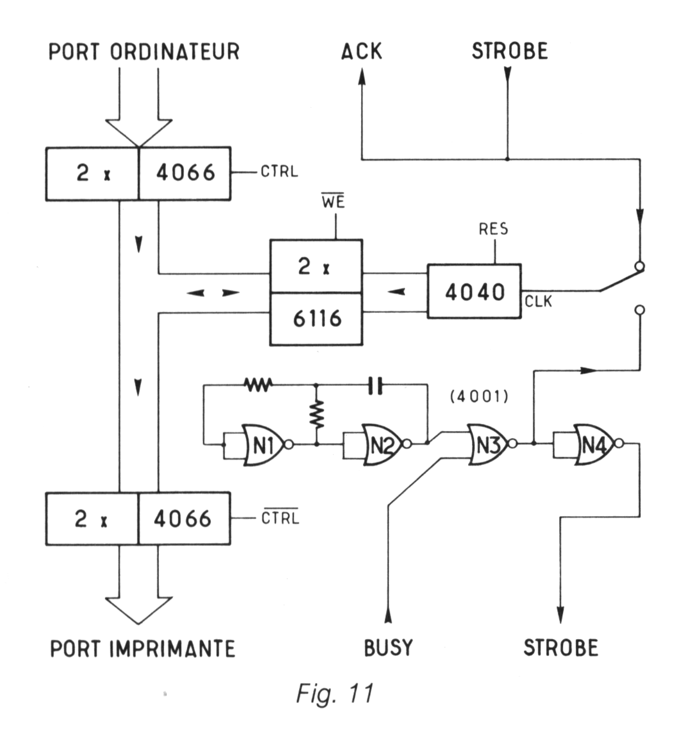


Fig. 11

n'est prévu pour arrêter le comptage. Il faudra terminer le bloc de données par un code désélectionnant ou bloquant l'imprimante.

Nous n'allons pas insister sur ce montage, qui reste entièrement théorique, contrairement aux extensions vues précédemment. Nous ne l'avons pas réalisé car plusieurs points nous sont apparus.

4 K c'est bien peu, cela ne peut guère servir qu'à stocker une ou deux pages de texte. En graphisme on aurait besoin de beaucoup plus, on pourrait étendre la capacité, mais les mémoires statiques coûtent cher. On penserait alors à employer des mémoires dynamiques, des 4164 par exemple, mais il faut les rafraîchir, ce qui compliquerait notre montage. Par ailleurs, on souhaiterait pouvoir utiliser cette mémoire plus librement, comme une véritable mémoire de masse externe. On aimerait de plus pouvoir la manipuler indépendamment de l'ORMOS, on songerait alors à y rajouter un CPU et du logiciel. Bref, on en vient à constater qu'il serait finalement bien plus simple, plus sûr, et sans doute moins onéreux, d'acheter un autre micro.

Et pourquoi pas ? Il n'est pas difficile de connecter deux ORMOS ensemble, ou un ORMOS à un micro d'un autre type, si l'on est curieux de toutes choses. Si ce micro ne possédait pas d'interface parallèle, il serait toujours possible d'en bricoler une. Toutes les familles de CPU présentent des boîtiers d'entrées-sorties conçus pour faciliter cette opération. La seule condition nécessaire est que le logiciel dispose des instructions PEEK et POKE ou d'un moniteur pour le langage machine.

On peut très bien connecter deux ORMOS via leurs VIA respectifs, pourvu que l'un au moins des appareils ait ses interruptions inhibées. Quant à la communication elle-même, c'est encore avant tout une question de protocole, mais on n'est pas limité dans ce cas par ce qui existe déjà, et l'on peut inventer son propre protocole en fonction des besoins.

On peut aussi très bien établir une communication sur 16 bits (plus 2 ou 4 lignes de contrôle) en montant deux CIA ou deux VIA. On transmettrait ainsi en une seule opération une information pouvant correspondre à une adresse.

On peut encore, plus protocolairement, réaliser une communication série à l'aide de deux ACIA et profiter ainsi d'un logiciel de transmission et d'acquisition de données pourvant servir à de multiples usages. Cette solution est à préconiser si l'on a déjà monté l'ACIA pour correspondre avec un quelconque périphérique, ou si l'autre micro est déjà doté d'une interface RS-232.

De délicats problèmes de préséance peuvent se poser, car il faudra définir un micro maître, et l'autre esclave...

20 La ROM en RAM et autres REM

Nous présentons ici quelques montages encore entièrement théoriques, ainsi que diverses idées.

• La ROM en RAM, c'est l'un des chemins de la liberté, remarquait l'auteur du RUM en un murmure amer. Amer, car à cette liberté s'associe une cécité totale.

Le SDEC 007 nous permettait déjà optionnellement cette opération, mais on peut la réaliser bien plus simplement avec un seul CI, en décodant uniquement les adresses 14-15 *(figure 12).*

L'inverseur S1 doit au départ être à la masse. On a ainsi accès à la ROM en lecture, et au bloc 3 de RAM en écriture. Il faut d'abord recopier la ROM en RAM par :

FOR A= #C000 TO #FFFF STEP 2 : DOKE A,DEEK(A) : NEXT

On a déjà des troubles à l'écran. On inverse alors S1, et on ne voit plus rien, mais on a toujours la main au clavier (si ce n'est pas le cas il faut faire un vrai RESET). On peut alors modifier la ROM en RAM à son gré, pourvu que ce soit pour une application sans l'écran. Nous n'avons trouvé qu'un moyen pour récupérer temporairement celui-ci, faire appel à une routine de temporisation en RAM normale, ou l'ULA lit la RAM en phase basse de PHI2. Les applications sont encore multiples, on peut par exemple transférer la ROM d'un Atmos en #4000-#7FFF, en faire une copie cassette, la charger sur Oric puis la transférer dans le bloc 3. L'intérêt est limité puisque l'on n'a pas d'affichage, mais qui sait ?

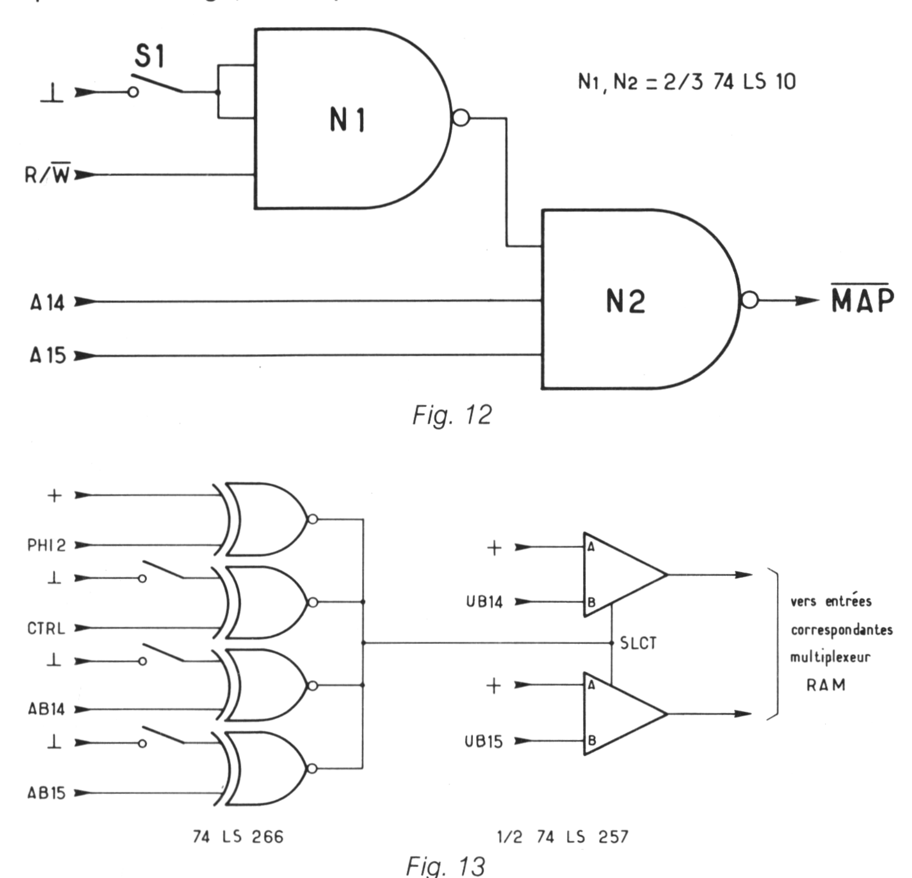


Fig. 12

Fig. 13

• Le montage de la figure 13 est dérivé du KGB. Le décodage des adresses 14 et 15 permet de placer le bloc 3 de RAM à l'emplacement du bloc 0, 1 ou 2 selon les presets et l'état du signal de commande. On peut ainsi, manuellement et théoriquement, permuter ces blocs de RAM. Nous n'insisterons pas non plus sur les multiples applications possibles.

• On peut connecter deux imprimantes sur le port parallèle à l'aide du montage de la figure 14.

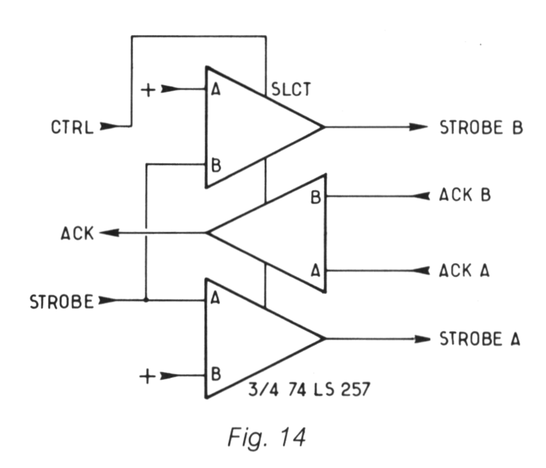


Fig. 14

Le signal de commande peut être fourni par un bit d'un CIA ou par toute autre source.

• On peut se servir de l'emplacement libre à côté de la ROM 16 K pour monter facilement une autre ROM. Une ROM 1-1 ou 1-0 par exemple, ou une EPROM programmée par vos soins avec vos petites routines secrètes. On veillera à ne pas souder la broche ROMDIS et à établir la liaison entre les deux CS. On sélectionnera l'un ou l'autre boîtier par son ROMDIS (l'un doit toujours être opposé à l'autre).

• Le 65 C 02 est un 6502 amélioré, entièrement compatible avec lui mais possédant un jeu d'instructions plus étendu. On peut donc déposer le 6502 de l'ORMOS et le remplacer par ce nouveau CPU. Seul problème, les assembleurs actuels ne reconnaissent pas les nouvelles instructions.

PROGRAMME BASIC EN RAM

Il est utile de savoir comment l'interpréteur Basic d'un système va ranger les données qui lui sont fournies. Une bonne assimilation de son fonctionnement peut permettre d'accroître considérablement les possibilités de l'ORMOS et de se délivrer de certaines lacunes ou contraintes du Basic.

Tout ce chapitre concerne non seulement l'Oric et l'Atmos, mais également beaucoup d'autres systèmes, le Basic de l'ORMOS se rapprochant fortement du standard Microsoft. Les Commodore, par exemple, possèdent exactement le même type de stockage des données. Les pointeurs auront bien sûr toutes chances d'avoir des emplacements différents.

Nous vous conseillons de procéder vous-même à des expériences et de vous livrer à de bonnes séries de DEEK et de PEEK pour consulter les contenus des pointeurs et des zones de mémoire programme et variables.

21 Pointeurs essentiels

Commençons par une brève présentation de Messieurs les Pointeurs Principaux. Ce sont des couples d'octets, le premier donnant la partie basse de l'adresse désignée, le second la partie haute. Ce sont les mêmes pour l'Oric et l'Atmos. Ils sont tous en page 0. Nous avons d'abord sept pointeurs consécutifs délimitant les zones accessibles au Basic :

— DB, adresses #9A-#9B (154-155) : c'est le début du programme Basic, il contiendra tant qu'on ne le modifiera pas #501 (1281), l'adresse de la première ligne de Basic ;
— DV, ou FBDV, #9C-#9D (156-157) : il contient l'adresse de la fin du programme Basic, celle à laquelle sera stockée une nouvelle ligne, ou la première variable. C'est donc aussi le début de la zone de stockage des variables ;

— DT, ou FVDT, # 9E- # 9F (158-159) : c'est la fin des variables et le début des tableaux ;
— FT, # A0- # A1 (160-161) : c'est la fin des tableaux, la première adresse disponible ultérieurement ;
— DC, # A2- # A3 (162-163) : c'est le début de la zone de stockage des chaînes de caractères ;
— UC, # A4- # A5 (164-165) : c'est un pointeur utilitaire qui présente peu d'intérêt pour l'utilisateur, il donne l'avant-dernière position du pointeur DC ;
— HM, # A6- # A7 (166-167) : c'est le HIMEM, la fin des chaînes, le premier octet inaccessible au Basic.

A l'initialisation, le pointeur DB est chargé avec la valeur # 501 (1281), les pointeurs DV, DT et FT sont chargés avec # 503 (1281 + 2). Les pointeurs DC et HM sont chargés avec la valeur de HIMEM, c'est-à-dire # 9 F00 (40704) pour l'Oric, et plus judicieusement # 9800 (38912) pour l'Atmos. En effet, les chaînes de caractères sont stockées à partir du haut de la RAM accessible alors que les variables le sont à partir du bas.

Un NEW va charger les pointeurs DV, DT et FT avec la valeur de DB augmentée de 2, et le pointeur DC avec la valeur de HM.

Un CLEAR, un RUN, un HIMEM ou une modification quelconque du programme vont repositionner les pointeurs DT et FT au niveau DV, et le pointeur DC au niveau HM.

Voici d'autres pointeurs qui concernent l'exécution même du programme :
— # A8- # A9 et # AA- # AB (168-169 et 170-171) : ils contiennent tous deux en programme le numéro de la ligne en cours d'exécution. Lors d'un arrêt du programme (par STOP, CTRL C ou RESET) l'adresse # A9 est chargée avec # FF, et DEEK(# AA) donne le numéro de la ligne à laquelle le programme a été interrompu, à moins que cette interruption ne résulte d'une erreur quelconque, auquel cas l'adresse # AB sera aussi chargée avec la valeur # FF. Un CONT sera alors inefficace. Ceci explique pourquoi un numéro de ligne ne peut être supérieur à 63999 (# F9FF) ;
— # AC- # AD et # E9- # EA (172-173 et 233-234) : ils sont positionnés tous deux en programme sur le prochain octet à interpréter. Lors d'un arrêt du programme, # AC- # AD garde l'information tandis que # E9- # EA revient au tampon clavier en # 35. # E9- # EA appartient en fait à la routine CHRGET implantée en page 0 à partir de # E2. Un DOKE # E9, DEEK(# AC) est l'équivalent d'un CONT. Si l'on veut positionner soi-même ces pointeurs, il faut leur donner l'adresse d'un zéro de fin de ligne ou d'un : séparant deux

instructions (CHR$(58)). DOKE # E9, 1280 est l'équivalent d'un RUN sans CLEAR;

— # B0- # B1 (176-177) : c'est le pointeur de datas utilisé par l'instruction READ. On peut par exemple ainsi faire un CLEAR sans RESTORE :

DOKE 0, DEEK(# B0) : CLEAR : DOKE # B0, DEEK(0)

— # B8- # B9 (184-185) : ce pointeur contient l'adresse de la dernière variable affectée. Il pointe en fait sur le premier octet représentant la valeur d'une variable entière ou réelle, ou sur l'octet contenant la longueur d'une variable chaîne de caractères. L'adresse du premier octet de cette chaîne est donnée par DEEK(DEEK(# B8) + 1). Notons que dans une expression du type A = f(X,Y,Z) ce sera toujours A la dernière variable affectée.

Précisons que ces pointeurs ne reflètent la réalité que lorsqu'ils sont utilisés uniquement par le système, c'est-à-dire par l'interpréteur Basic. Le programmeur a accès à ces pointeurs, mais il ne faut bien sûr modifier leur contenu que dans un but précis et à bon escient. Nous en saurons bientôt assez pour ce faire.

22 Stockage du programme Basic

A l'initialisation, nous avons donc BD ou DEEK(# 9A) égal à # 501 (1281), et DV égal à # 503. Si l'on fait quelques PEEK (dumping) à partir de 1280, on trouve :

(1280) : 0 0 0 85 85 85 et pas mal d'autres 85

85 (01010101 en binaire) est la valeur chargée dans toutes les cases de RAM par une routine de vérification à l'initialisation. Entrons alors une ligne telle :

10 HIMEM 12345

Le pointeur DV vaudra maintenant # 50F (1295), et l'on va trouver en mémoire :

(1280) : 0 13 5 10 0 158 32 49 50 51 52 53 0 0 0 85 85 et des 85 à la pelle.

Le couple 1281-1282 est le lien, c'est une valeur hexa sur 2 octets qui vaut ici # 50D ou 1293. Il pointe ici sur le deuxième 0 de la série de trois 0 avant les 85. Les 2 octets suivants sont chargés avec la valeur hexa du numéro de ligne (de 0 à 63999). Ensuite on trouve le contenu de la ligne. 158 est le code de HIMEM, 32 le code ASCII de l'espace, 49 à 53 les codes ASCII respectifs des caractères « 1 » à « 5 ». Le 0 suivant est un code de fin de ligne, les deux 0 suivants

signifient la fin du Basic. C'est à leur place que viendra le prochain lien. Le lien pointe toujours vers le lien suivant. Ces liens permettent à l'interpréteur de parcourir très rapidement tout le programme sans s'attarder au contenu des lignes ou à leur longueur.

Résumons-nous. Lorsque l'on tape une ligne au clavier, les caractères sont stockés l'un après l'autre dans un tampon de 80 octets débutant en #35 (53). Après un RETURN, l'interpréteur examinera ce tampon et procédera aux opérations suivantes, non obligatoirement dans cet ordre :

— il identifiera les mots clés et les remplacera par des codes de 128 à 255, par un seul octet donc ;
— il consultera les pointeurs DB et DV ;
— il s'assurera que le programme situé entre ces adresses ne comporte aucune ligne de numéro supérieur ou égal à celui de la ligne en cours d'interprétation ;
— il stockera à l'adresse donnée par DV le numéro de ligne, sur 2 octets ;
— il recopiera la ligne compilée dans les cases suivantes ;
— il écrira ensuite un 0 de fin de ligne, puis deux 0 de fin de programme ;
— il calculera le lien pointant sur le premier de ces deux 0 et le stockera aux adresses DV-2 et DV-1 ;
— il remettra à jour les pointeurs DV, DT et FT en leur donnant pour contenu l'adresse de l'octet suivant les deux 0 de fin de programme.

C'est le cas le plus simple. Si la nouvelle ligne vient s'intercaler entre deux lignes déjà existantes, l'interpréteur calculera la longueur de la ligne à insérer, puis il décalera vers le haut tout le bloc de RAM depuis la ligne de rang immédiatement supérieur jusqu'à l'adresse donnée par le pointeur DV. Il recalculera tous les liens, stockera la nouvelle ligne dans l'espace aménagé, et remettra les pointeurs à jour.

On déduira aisément de ce que l'on sait de l'écriture des programmes le comportement de l'interpréteur lorsqu'il doit traiter une ligne dont le numéro existe déjà, ou dont le numéro est inférieur à celui de la première ligne du programme déjà en mémoire.

23 Stockage des variables

Les variables ordinaires, c'est-à-dire non indicées, sont stockées sur 7 octets, quel que soit leur type, réelles, entières, ou chaînes de caractères.

Les deux premiers octets représentent le nom de la variable. Ils sont codés comme suit :

— pour un nombre réel : code ASCII première lettre et code ASCII deuxième caractère. Ainsi A1 sera codé #41 #31 (65 49) ;
— pour une variable entière : code ASCII première lettre + 128 et code ASCII deuxième caractère + 128. A1 % sera codé #C1 #B1 (193 = 65 + 128, 177 = 49 + 128) ;
— pour une chaîne de caractères : code ASCII première lettre et code ASCII deuxième caractère + 128. A1$ sera codé #41 #B1 (65 177 = 49 + 128) ;

Si le nom de la variable ne comporte qu'une seule lettre, le code ASCII du second caractère sera évalué à 0.

Seules les variables réelles ont réellement besoin des 5 octets restants pour leur représentation, mais toutes les variables sont dimensionnées sur 7 octets pour simplifier la tâche de l'interpréteur Basic, qui pourra ainsi les consulter successivement par bonds toujours identiques.

Nous n'allons pas détailler la représentation des variables réelles, qui n'est pas facilement exploitable directement par le programmeur. L'important est de connaître leur emplacement.

Les variables entières sont stockées sur 2 octets. A l'inverse de ce qui est habituel, le premier octet est celui de poids fort. De plus il s'agit d'un nombre signé. S'il est supérieur à #7FFF il vaudra sa valeur hexa — #10000 (65536). Par exemple #7FFF vaut —1. Trois 0 seront placés dans les 3 octets suivants inutilisés.

Les chaînes de caractères auront besoin de 3 octets, le premier représentant la longueur de la chaîne (LEN(X$)), les deux octets suivants pointant sur l'adresse du premier octet à partir de laquelle elle est stockée. Deux 0 seront placés dans les deux octets suivants pour complémenter la longueur de la variable à 7 octets.

Mais où sont donc stockées ces chaînes ? Deux cas se présentent :

— si la chaîne fait partie d'une ligne de programme sous la forme X$ = « XXXX », l'interpréteur ne se donnera pas la peine de gaspiller des octets toujours précieux pour la stocker ailleurs. Il affectera aux deux octets concernés de la variable X$ la valeur de l'adresse à partir de laquelle est stockée la chaîne à l'intérieur du programme ;
— dans les autres cas, lorsque la chaîne est le résultat d'une opération quelconque, ou lorsqu'elle est entrée directement au clavier, elle sera stockée en haut de la mémoire à partir de l'adresse DEEK(#A2)-LEN(X$). Le pointeur DC se verra chargé avec cette

nouvelle valeur, ainsi que les octets 4 et 5 de la variable X$. Le pointeur UC se verra chargé avec l'ancienne valeur de DC.

Examinons maintenant ce qui se passe lorsque l'interpréteur rencontre une variable non indicée :

— il va coder le nom de la variable selon sa nature ;
— il va examiner dans la zone DV à DT s'il existe déjà une variable de ce nom ;
— s'il trouve cette variable, il lui affectera sa nouvelle valeur ;
— sinon il décalera la zone des tableaux de DT à FT de 7 octets vers le haut et inscrira dans l'espace libéré la nouvelle variable ;
— il remettra dans ce cas les pointeurs à jour (DT = DT + 7 et FT = FT + 7).

Si de plus la variable est une chaîne de caractères, cette chaîne sera stockée comme énoncé plus haut. A noter que si l'on affecte successivement plusieurs chaînes à une même variable, toutes seront stockées les unes après les autres, ou plutôt les unes avant les autres. Ainsi l'interpréteur n'aura pas à modifier les adresses des autres chaînes présentes dans les variables. Cette modification ne se produira que lorsque les chaînes viendront se casser le nez contre le pointeur FT, ou après un ordre FRE(0). Après un FRE(0), l'interpréteur va passer en revue toutes les variables chaînes présentes en mémoire, y compris les tableaux, et il va restocker toutes les chaînes correspondantes à partir du pointeur HM, en notant bien sûr soigneusement les nouvelles adresses dans chaque variable.

24 Stockage des tableaux

Que se passe-t-il maintenant lorsque l'interpréteur rencontre une variable indicée ? Rappelons que si un tableau n'a pas été préalablement dimensionné, il le sera automatiquement lorsque l'interpréteur rencontrera un de ses éléments, à condition qu'aucune de ses dimensions ne soit supérieure à 10. Le tableau sera stocké à partir de DT, et si ses dimensions n'ont pas été précisées par une instruction DIM, elles se verront arbitrairement attribuer la valeur 10. Si nous entrons par exemple A1(1,2,3) = 0, voici ce que nous trouverons en mémoire, à partir de l'adresse contenue en DT :

65 49 10 26 3 11 0 11 0 11 0 puis beaucoup de 0.

Les 2 premiers octets représentent le nom du tableau, ils ont été codés exactement selon les mêmes principes que ceux utilisés pour les variables non indicées. Les 2 octets suivants contiendront la

valeur hexa de la taille du tableau, soit ici # 1A0A ou 6666 octets. L'octet suivant contient le nombre de dimensions, puis on trouve autant de groupes de 2 octets qu'il y a de dimensions, chacun contenant la valeur hexa du nombre d'éléments de chacune d'entre elles. Ici on a trois 11 car une dimension d'ordre 10 contient 11 éléments, l'ORMOS ne possédant pas l'option BASE 1. Viendront ensuite les 11*11*11 éléments du tableau, soit 1331 groupes de 5 octets puisqu'il s'agit de variables réelles. On a bien :

$$2 + 2 + 1 + (2 \times 3) + (5 \times 1331) = 6666 \text{ octets}$$

Imaginons qu'en fait nous n'avions besoin que de $2 \times 3 \times 4$ éléments et qu'il s'agisse de plus d'entiers. Entrons DIM A1%(1,2,3) et allons voir ce qui ce passe en DT. Nous trouverons cette fois les valeurs suivantes :

193 177 59 0 3 2 0 3 0 4 0
puis beaucoup moins de 0 que dans l'exemple précédent.

On a en effet : $2 + 2 + 1 + (2 \times 3) + (2 \times 2 \times 3 \times 4) = 59$ octets.

Ceci devrait suffire à démontrer qu'on a souvent intérêt à dimensionner ses tableaux, et que les variables entières peuvent trouver leur utilité.

Nous pouvons donc établir une formule générale pour évaluer le nombre d'octets occupé par un tableau, ce sera :

Taille tableau = $5 + (ND \times 2) + (NE \times X)$ avec
ND = Nombre de Dimensions
NE = Nombre d'Eléments total
X = 2, 3 ou 5 selon variable entière, chaîne ou réelle.

Sans se casser la tête, on peut aussi noter FT, dimensionner le tableau, demander la nouvelle valeur de FT et faire la différence.

Venons-en maintenant à la situation d'un élément dans le tableau. Dans l'exemple précédent on trouvera dans cet ordre les éléments : A1%(0,0,0), A1%(1,0,0), A1%(0,1,0), A1%(1,1,0), A1%(0,2,0), A1%(1,2,0), A1%(0,0,1), A1%(1,0,1),...

On aura compris que c'est la première dimension qui joue le rôle des unités, la seconde celle des « dizaines », etc.

Par ailleurs, on peut toujours trouver en Basic l'adresse d'une variable indicée ou non à l'aide du pointeur # B8- # B9, il suffit d'entrer une instruction neutre telle PRINT A ou A(3) = A(3) puis de faire ? DEEK(# B8).



La connaissance du fonctionnement des pointeurs *(V. 21 et suivants)* va nous permettre d'intervenir directement sur le programme Basic et sur les variables. Les possibilités sont immenses et nous ne saurions donner ici que quelques idées.

25 Renuméroter

Voici un sous-programme tout à fait classique, qui permet de bien comprendre le rôle des liens.

```
60000 INPUT"Début de la renumérotation";A
60010 INPUT"Fin de la renumérotation";E
60020 INPUT"Début des nouvelles lignes";D
60030 INPUT"Incrément";C
60040 L=DEEK(#9A)
60050 N=DEEK(L+2)
60060 IF N>E THEN END
60070 IF N<A THEN 60090
60080 DOKE L+2,D : D=D+C
60090 L=DEEK(L) : GOTO 60050
```

Attention, il faudra recalculer les GOTO, GOSUB, THEN et autres ELSE.

26 Reculer le début du Basic et se ménager une zone tranquille en RAM

La procédure est très simple ; si DB doit être le nouveau début du Basic, il suffit de faire :

DOKE #9A,DB : POKE DB-1,0 : NEW

Le 0 en DB-1 est nécessaire à l'interpréteur. Le NEW va mettre deux 0 en DB et DB + 1 et charger les pointeurs DV, DT et FT avec DB + 2.

On peut récupérer la page 4 pour le Basic en faisant DB = 1025 (valeur minimale de DB). Pratiquement cela n'a guère d'intérêt. Cette page 4 sert le plus souvent à stocker quelques routines machine. Il se peut que 256 octets s'avèrent insuffisants, ou que l'on ait besoin d'y caser un fichier de données. La zone située en dessous du Basic a l'avantage de lui être adjacente, par opposition à une autre zone protégée, au-dessus du HIMEM par exemple. On peut ainsi la sauvegarder en même temps que le texte Basic par un CSAVE A1024. Il faudra cependant réajuster le pointeur DB avant l'exécution du programme. On peut éviter cet inconvénient en créant une zone à l'abri du Basic à l'intérieur même du programme Basic. Voici la marche à suivre :

Admettons que l'on ait besoin de 9 000 octets environ. On reculera d'abord le début du Basic à 10 000 par :

DOKE #9A,10000 : POKE 9999,0 : NEW

On écrira ensuite le programme, tout à fait normalement. Notons qu'il est ici possible de charger directement un programme sur cassette en utilisant l'option J de l'Atmos. Lorsque ce programme sera en mémoire, et qu'il n'y aura plus de retouches à y apporter, on fera :

DOKE 1281,10000:DOKE 1283,0:DOKE 1285,157: DOKE #9A,1281

Ce qui revient à ramener le début du Basic à sa position habituelle et à écrire la ligne 0 REM, le lien de celle-ci pointant vers le programme entré précédemment. On peut ainsi lister, sauvegarder et exécuter normalement un programme Basic comportant une zone non Basic. Il n'est toutefois pas possible de modifier directement ce programme, car l'interpréteur recalculerait en ce cas le premier lien. Il faudrait avant toute modification ramener le DB à son niveau modifié par un DOKE #9A, 10000, puis revenir à la valeur normale une fois le programme corrigé (DOKE #9A,1281).

Les possibilités d'utilisation de cette zone sont multiples, langage machine, fichier, sauvegarde d'écrans...

27 Retrouver programme et variables après un NEW

Ce cas est extrême et assez rare, il s'agit en fait de pouvoir rétablir des pointeurs perturbés pour une raison quelconque. Pour récupérer le programme Basic, il faut restaurer le lien en 1281 et le pointeur de fin du Basic. On suppose ici que le Basic commence effectivement en 1281. La première opération s'effectuera en détectant le premier zéro de fin de ligne ; on fera en mode direct :

```
DOKE 2,1285
REPEAT:DOKE 2,DEEK(2)+1:UNTIL PEEK(DEEK(2))=0
DOKE 1281, DEEK(2)+1
```

On n'emploie aucune variable pour éviter toute modification de la zone Basic. On utilise la même méthode pour détecter les trois zéros significatifs de la fin du Basic :

```
DOKE 2, 1285
REPEAT:DOKE 2,DEEK(2)+1:UNTIL DEEK(DEEK(2))
+ PEEK (DEEK(2)+2) = 0
DOKE #9C, DEEK(2)+3
```

Si le bouleversement des pointeurs ne provient pas d'un NEW, il faut vérifier que DB est bien à 1281 et HIMEM supérieur à FB. On peut alors lister le programme ou le lancer par un RUN.

Si l'on a perdu les pointeurs des zones variables, après CLEAR, RUN ou HIMEM par exemple, on peut les recalculer relativement aisément puisque l'on peut lister le programme. On suppose néanmoins que le début des variables correspond à la fin du Basic *(V. 30)*. La valeur primordiale à retrouver est FV (Fin des Variables). On peut la calculer ainsi :

$$FV = DV + NV \times 7$$

avec NV = Nombre total des Variables non indicées.

Ce peut ne pas être évident s'il y a beaucoup de variables ou si leurs affectations sont incertaines. On peut procéder par tâtonnements en consultant les octets contenant les noms des variables pour débusquer un éventuel tableau. De toute manière, s'il n'y a pas de tableau, on peut sans dommage dimensionner largement FV, pourvu que FV — DV soit un multiple de 7.

S'il y a des tableaux, on vérifie qu'ils ont bien été dimensionnés (pour AA(X) par exemple, on doit avoir DEEK(FV) = #4141). Admettons qu'il n'y ait qu'un seul tableau, on a alors FT = FV + DEEK(FV+2). Il suffit de répéter l'opération s'il y en a plusieurs.

On peut ensuite restaurer les pointeurs en faisant :

DOKE #9E, FV : DOKE #A0, FT : ? FRE(0)

Le FRE(0) permet de restaurer le début des chaînes (DC).

FB, FV et FT doivent être fournis sous forme numérique avant la restauration des pointeurs si l'on veut éviter tout risque de perturbation.

28 Retoucher un programme sans perdre les variables

Ce n'est nullement impossible, pourvu que l'on n'écrive pas de ligne de numéro supérieur à celui de la dernière du programme. Voici la marche à suivre :

— noter les valeurs de DV, DT, FT ;
— doker la valeur de FT en DV et DT ;
— effectuer les modifications désirées ;
— consulter DT, lui soustraire la première valeur de DT préalablement notée ;
— doker en DV et DT les valeurs respectives augmentées du résultat de cette soustraction (ce résultat peut être négatif si l'on a supprimé des lignes).

Pratiquement on peut faire :

DOKE 2,DEEK(#9C):DOKE 4,DEEK(#9E): DOKE 6,DEEK(#A0)
DOKE #9C,DEEK(6):DOKE#9E,DEEK (6)

Retoucher le programme :

DOKE 8, DEEK(#A0)
DOKE #9C,DEEK(2)+DEEK(8)-DEEK(6)
DOKE #9E,DEEK(4)+DEEK(8)-DEEK(6)

Il est éventuellement possible de rentrer ces instructions sous la forme de deux sous-programmes.

29 Relancer le programme après une erreur ou un RESET

Après un RESET on a le numéro de ligne en cours en #AA-#AB. Un message d'erreur nous donne en principe le numéro de la ligne incriminée. Par ailleurs DEEK(#AC) donne l'adresse du prochain octet

à interpréter. Nous venons de voir que nous pouvons rectifier un programme sans perdre les variables, en conséquence il doit être possible de le relancer par un GOTO une fois éliminées les causes d'erreur, excepté lorsque le programme est en cours d'exécution d'instructions FOR NEXT, REPEAT UNTIL ou GOSUB. Ces instructions utilisent la pile du 6502 en page 1, qui est réinitialisée après un RESET, une erreur ou une modification du programme. Notons que cette réinitialisation peut avoir lieu pendant un BREAK tout à fait légal si un quelconque message d'erreur apparaît pendant ce BREAK, et qu'un fatidique CAN'T CONTINUE ERROR peut sanctionner une simple erreur de syntaxe ou un mauvais argument en mode direct.

On peut néanmoins relancer un programme interrompu au cours d'une boucle en rentrant la boucle en mode direct. Si l'on a par exemple un programme du type :

```
500 FOR A=1 TO 100
510 Première instruction
520 Suite...
800 NEXT
```

et que l'on a eu un problème pour A = 50, et que la relance du programme au début demande trop de temps ou de travail, il est possible de sauver la situation, une fois le problème résolu, en entrant en mode direct :

```
FOR A=50 TO 100 : GOTO 510 : NEXT
```

Le programme continuera alors normalement jusqu'à la ligne 800 qui provoquera un NEXT WITHOUT FOR ERROR, mais on sera sorti de la boucle et on pourra le relancer par un GOTO 810 par exemple. Pour une boucle REPEAT on ferait REPEAT : GOTO ligne : UNTIL condition jusqu'au BAD UNTIL. Par contre on ne peut employer GOSUB en mode direct.

30 Choisir une zone variables

Le pointeur DV est véritablement le pointeur de début des variables. On peut le modifier et lui donner une valeur plus élevée sans observer de modifications dans le listage ou l'exécution du programme, pourvu qu'on ait réajusté les autres pointeurs à son niveau par un CLEAR. La fin du Basic est signifiée à l'interpréteur par les trois zéros de fin de ligne et de lien nul. Quel est l'intérêt de cette possibilité ? On peut par exemple fixer la valeur de DV pour un programme

appelé à subir des modifications. On le fera en principe en première ligne par un DOKE #9C, DV. Ceci permet non seulement de savoir exactement où seront les variables, mais encore de transmettre ces variables à un autre programme possédant le même DV *(V. 50, 51).*

On peut suivre des procédures analogues avec le pointeur DT. Rappelons qu'il est assez impératif que DT-DV soit un multiple de 7.

Il est encore possible de donner plusieurs valeurs successives aux pointeurs DV, DT et FT à l'intérieur d'un même programme. On peut ainsi délimiter plusieurs zones de variables, auxquelles on peut accéder à tout moment en rétablissant les valeurs correspondantes des pointeurs. On a ainsi des variables variables... Le principal intérêt est la possibilité de passer très rapidement, par quelques DOKE, d'un jeu de variables à un autre, au lieu de devoir leur réaffecter individuellement de nouvelles valeurs.

Il serait encore possible, sous certaines conditions *(V. 26)*, de placer la zone variables à l'intérieur même du programme Basic, de manière à pouvoir sauvegarder un jeu de variables avec ce programme.

31 Fusionner des lignes de Basic

La longueur d'une ligne de Basic est limitée en mode direct par la taille du tampon clavier, soit 80 octets. Pratiquement cette longueur sera réduite à moins de 80 caractères affichables, selon le nombre de chiffres du numéro de ligne et la longueur des mots clés qui ne seront codés que sur un seul octet. En fait l'interpréteur accepte des lignes comportant jusqu'à 250 octets, c'est-à-dire nettement plus de caractères affichables. La limitation provient de ce que la différence entre deux liens consécutifs ne peut être supérieure à 255, excepté dans certaines conditions *(V. 26).*

Voici quelques lignes de Basic qui permettent d'opérer la compilation de lignes ultérieures, jusqu'à cette valeur de 250 octets (255 = 250 + 2 octets du lien + 2 octets du numéro de ligne + 1 octet nul) :

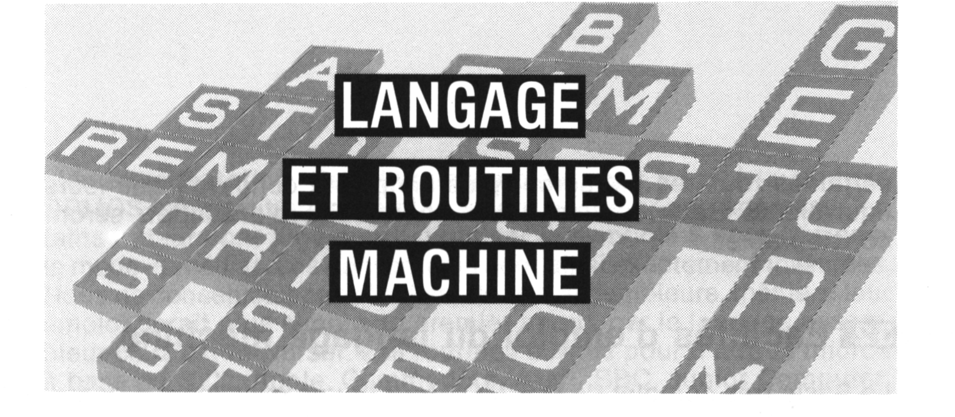
Après avoir rentré ces instructions, il faudra faire un GOTO 10. On entrera ensuite deux lignes à compiler, qui devront avoir des numéros supérieurs à tout numéro déjà existant, et on fera un GOTO 20. Les deux lignes seront alors remplacées par une seule. Il suffira d'en entrer une nouvelle et de refaire GOTO 20 pour ajouter les instructions de cette ligne à la précédente, ceci jusqu'à concurrence de

250 octets. Lorsque l'on voudra recommencer l'opération pour une autre ligne, on fera un GOTO 10 avant d'entrer les nouvelles lignes à compiler...

Cette compilation permet d'accélérer la vitesse d'exécution d'un programme. Elle permet dans certains cas de supprimer des instructions (IF THEN ELSE à instructions multiples par exemple).

```
0 REM Compilation de lignes BASIC
10 DOKE 2,DEEK(#9C) : STOP
20 A=DEEK(2)-2 : B=DEEK(A)
30 POKE B-1,58 : DOKE A,DEEK(B)-4
40 FOR C=B+4 TO DEEK(#9C)-1:POKE C-4,PEEK(C):NEXT
50 DOKE #9C,DEEK(#9C)-4 : CLEAR : STOP

100 REM Exemple de ligne compilée. On a fait GOTO 1
0, puis entré 100 jusqu'ici:REM puis entré une lign
e 110 commençant par REM, et enfin fait GOTO 20
```



LANGAGE ET ROUTINES MACHINE

La programmation en langage machine, LM en abrégé, s'imposera si l'on veut vraiment tirer le maximum de son appareil. Le premier avantage du LM sera sa vitesse, sans commune mesure avec celle du Basic. Les rapports sont en fait très variables, car les instructions du Basic appellent des routines en LM, seul langage que peut comprendre le 6502, mais ces routines sont conçues pour des applications très générales. Une opération telle que A = A × 2 sera effectuée de la même manière qu'une multiplication de deux variables réelles, ce qui peut demander plusieurs centaines d'instructions élémentaires, alors qu'une seule instruction, occupant un seul octet, fera la même chose, pourvu que A soit inférieur à 128. De plus le Basic n'aura pas à interpréter un par un les caractères d'une ligne, ce qui prend également beaucoup de temps.

Il va de soi que l'on ne se lancera pas au départ dans la réalisation d'un programme entier en LM. On se contentera d'écrire des routines, le plus souvent extrêmement simples, ne comportant qu'un nombre réduit d'instructions élémentaires, qui seront appelées par le programme Basic. Diverses routines de modification de pointeurs, par exemple, peuvent être immédiatement traduisibles en LM, ce qui ne peut qu'améliorer la présentation d'un programme Basic.

Nous n'allons pas détailler ici le jeu d'instructions du 6502, ni ses 13 modes d'adressage. Un livre entier est pratiquement nécessaire pour cela, et de bons ouvrages existent déjà. La maîtrise totale du LM n'est certes pas chose aisée, mais il en va de même du Basic. Et de même qu'il est possible de programmer en Basic avec un jeu très limité d'instructions, on peut se limiter au départ en LM à quelques instructions fondamentales et n'employer que quelques modes d'adressage. De plus, le LM possède certaines instructions très puissantes qui n'ont aucun équivalent direct en Basic. En conséquence

certaines routines LM seront plus simples à concevoir que leurs équivalents Basic, et les gains en rapidité d'exécution seront considérables.

32 Facilités d'emploi du langage machine

Le Basic très complet de l'ORMOS fournit une excellente préparation au LM. Les fonctions PEEK, POKE, OR, AND par exemple ont leurs équivalents directs en LM. De fait on peut souvent traduire directement un programme Basic en LM en respectant sa structure logique. Ce n'est pas toujours la meilleure solution, mais une routine LM même fort mal conçue aura toutes chances d'être exécutée bien plus vite que son meilleur équivalent Basic.

La possibilité en standard d'employer l'hexadécimal est aussi un très gros atout. On peut ainsi charger directement du code machine à l'aide de quelques lignes de Basic par exemple *(V. 58)*. Toutefois les risques d'erreurs sont assez grands. Il faut faire extrêmement attention aux B et 8 qu'il est très facile de confondre. Et même si l'on arrive à recopier une floppée de code hexa sans erreur, encore s'agit-il d'être certain que ce que l'on recopiait était bien exempt d'erreur. Il est tout à fait recommandé d'utiliser au moins un assembleur pour éviter de tels problèmes. On ne charge ainsi plus de codes hexa dont il est difficile de se rappeler la signification, mais des mnémoniques faciles à retenir. De plus la structure logique du programme est apparente, ce qui permet d'éviter les erreurs sur les opérandes. Encore faut-il que l'auteur du programme en ait fourni un listing désassemblé, ce qui n'est pas toujours le cas, souvent pour une question de place. La checklist est alors un bon moyen de vérification (pourvu qu'elle soit bonne), mais il n'est pas du tout improbable de prendre une fois un B pour un 8, et une autre fois l'inverse.

Un désassembleur permettra de vérifier que le programme rentré en hexa correspond bien à une suite d'instructions.

Un assembleur-désassembleur est donc un outil fort utile pour la programmation en LM. Il se pose toutefois un problème de syntaxe. Il existe des conventions très généralement employées dans la syntaxe assembleur 6502. Le symbole $ est employé pour désigner une valeur hexa. Employé seul il représente une adresse : LDA $02 signifie charger l'accumulateur avec le contenu de l'adresse $2. Précédé de #, $ représente une valeur immédiate : LDA #$02 signifie charger l'accumulateur avec la valeur $02. On sait que dans la syntaxe ORMOS # désigne les valeurs hexa, il y a donc ici un pro-

blème d'adaptation. Certains assembleurs ont adopté la syntaxe ORMOS, omettant en général le # pour désigner une adresse. Certains auteurs ont jugé subtil d'utiliser le symbole $ pour désigner le mode immédiat. Les conventions sont ainsi exactement inversées. Nous ne conseillons pas d'utiliser de tels assembleurs, même si leur emploi paraît plus simple au premier abord, car le langage assembleur 6502 est universel. Les routines écrites pour d'autres micros à base de 6502 (Apple, Commodore, Atari, BBC, Junior Computer, Kim) seront transposables sur ORMOS, et la syntaxe employée sera souvent la plus conventionnelle.

Les routines présentées dans cet ouvrage ont été écrites et listées à l'aide du moniteur 1-0 de Loriciels (il existe une version compatible 1-1), qui utilise cette syntaxe universellement admise. Une ligne de listing correspondant à une instruction élémentaire, elle se présentera comme suit :

```
0400: A5 04 LDA $04
```

On aura d'abord l'adresse du premier octet de l'instruction, puis le code machine sous forme d'1, 2 ou 3 octets en hexa, puis le mnémonique de l'instruction suivi s'il y a lieu d'un opérande. Cet opérande sera donc précédé de $ s'il désigne une adresse, et de # $ s'il représente une valeur immédiate.

Notons qu'un ORMOS fonctionnant sous le contrôle d'un moniteur n'est plus un ORMOS standard. En conséquence une routine conçue pour être intégrée à un programme Basic peut ne pas se comporter comme prévu tant que le moniteur sera présent en mémoire.

L'ORMOS dispose par ailleurs de grandes facilités pour écrire et appeler des routines en LM.

de nombreuses adresses libres en page 0.

Les 12 premières adresses au moins (de # 00 à # 0B) sont inutilisées par le système en configuration standard. Elles sont donc à la disposition du programmeur. Il en existe quelques autres. On peut par ailleurs se servir momentanément ou non selon les programmes des 80 octets du tampon clavier (de # 35 à # 84).

la page 4

Cette page est visiblement prévue pour l'écriture de routines LM. Son avantage est d'être contiguë au Basic. On peut donc la sauvegarder en même temps qu'un programme Basic par un CSAVE"", A # 400 (ou A 1024). Sur Atmos il faudra faire après une relecture DOKE

9C, DEEK(# 2AB) car le programme sera supposé être entièrement du code machine.

Il y a moyen de faire pas mal de choses avec 256 octets, mais on peut rallonger la sauce en reculant le début du Basic *(V. 26).* Il faut alors sauvegarder quelque part le pointeur DB (# 9A).

d'autres zones intéressantes

Les tables de caractères présentent d'intéressantes propriétés. La zone # B400 à # BAFF qui les contient est recopiée lors d'un HIRES aux adresses # 9800 à # 9FFF. Cette zone est à son tour recopiée en # B400-# BAFF lors d'un retour en mode TEXT, mais le contenu de # 9800-# 9FFF restera alors inchangé pourvu que le HIMEM ne dépasse pas # 9800. Or les pages # B4 et # B8 (et les pages correspondantes # 98 et # 9B) ne servent pas réellement au générateur de caractères et ne sont pas affectées par la restauration des tables, par un RESET par exemple. On peut donc encore récupérer pour le LM deux pages normalement inaccessibles au Basic, étant bien compris qu'il faudrait le plus souvent charger une routine destinée à servir en HIRES en page # B8 pour pouvoir la recopier en page # 9B par un HIRES. On peut caser aussi du LM dans l'une ou l'autre des tables de caractères (ou dans les deux), mais ces tables sont réinitialisées par un RESET.

On a encore bien d'autres possibilités pour caser du LM, abaisser le HIMEM, l'inclure au sein du Basic *(V. 26)* ou utiliser les 16 K de RAM normalement masqués par la ROM *(V. 17, 18, 20).*

les instructions CALL et !

La richesse du Basic de l'ORMOS est ici évidente. CALL est bien connu, mais on dispose en outre de !, qui est équivalent à un CALL DEEK(# 2F5). Donc si l'on doit appeler plusieurs fois dans un programme une même routine on rangera son adresse en # 2F5 par un DOKE. On peut appeler élégamment plusieurs routines en faisant suivre le ! d'un autre caractère, dont le décodage sera effectué au point d'entrée commun des diverses routines. Il faudra incrémenter le pointeur # E9-# EA avant de revenir au Basic (adresse du prochain octet à interpréter).

les instructions USR(X) et &(X)

Encore deux instructions équivalentes. Elles appellent les routines dont les adresses sont rangées respectivement en # 22-# 23 (34-35) et # 2FC-# 2FD (764-765). On dokera donc les adresses voulues dans

ces vecteurs avant d'appeler les routines. Le paramètre ou la variable X est transféré dans l'accumulateur flottant ACC1, situé aux adresses # D0 à # D5. Nous verrons plus loin le rôle de cet accumulateur *(V. 34).*

33 Similitudes et divergences

Tous les chemins mènent à la ROM, mais laquelle ? Il est souvent bien utile d'appeler des routines ou des sous-routines en ROM plutôt que de les réécrire ou de revenir au Basic, mais nous touchons ici aux limites de la compatibilité. Bien que la plupart de ces routines soient absolument identiques sur les deux systèmes, leurs adresses sont différentes. On peut néanmoins exploiter certains points.

— # C000 : un CALL # C000 correspond à un RESET assez tiède. Il ne restaure cependant pas les vecteurs d'interruption en RAM, mais il réinitialise toutes les autres adresses stratégiques. Il appelle le NEW, mais le programme est récupérable *(V. 27) ;*

— # C003 donne le genre RESET brûlant, il ne réinitialise en fait pas grand chose ;

— de # C006 à # C0E9 se trouve une table des adresses des points d'entrées des routines correspondant aux mots clefs du Basic. C'est très intéressant, car cette table obéit aux mêmes lois dans les deux ROM. En bref, si le code correspondant à une instruction Basic est C, on trouvera l'adresse de la routine correspondante par la formule :

A = HEX$(DEEK(# C006 + (C-128)*2) + 1[e]

Cette formule n'est en fait pas rigoureusement valable pour tous les mots clefs, mais elle est utilisable pour toutes les instructions ne demandant aucun paramètre, comme HIRES, TEXT, CLEAR, RUN... On peut ainsi appeler ces fonctions par un même programme LM pour les deux systèmes. Pour les autres fonctions, il faudra ruser, établir des tables des routines utilisées, faire un test en ROM initialisant l'une ou l'autre table par exemple ;

— à partir de # C0EA se trouve une table des mots clefs correspondant aux codes 128 à 255. La fin d'un mot clef est signifiée par le bit 7 du dernier caractère à 1 ;

— une vaste zone vient ensuite, où toute correspondance ne saurait être que le fruit du hasard. Toutefois, lorsque l'on connaît l'adresse d'une routine sur un système, il y a de grandes chances de trouver la routine correspondante dans la même zone de l'autre ROM ;

— de # FC70 à # FF6F se trouve la table des caractères standard, dans l'ordre selon lequel elle sera recopiée en # B500. On a sur l'Atmos un caractère supplémentaire, de code 127 ;
— suit une table d'ASCII décalée de 8 octets sur l'Atmos ;
— enfin on a les vecteurs hardware. Si leurs contenus sont différents on dispose néanmoins de DEEK(# FFFA), DEEK(# FFFC) et DEEK(# FFFE) qui représentent les mêmes fonctions sur les deux systèmes, NMI, RES et IRQ *(V. 4, 44).*

Les vecteurs NMI et IRQ nous renvoient donc en RAM, à des adresses différentes il est vrai, mais notre principal problème est ailleurs. L'Atmos possède quatre autres détours en RAM qui n'ont aucune équivalence sur l'Oric. Il semble évident qu'il n'y a aucune solution simple permettant d'adapter un programme Atmos qui détournerait l'une de ces quatre routines.

Pour le reste la compatibilité est excellente. Les pointeurs et variables-système occupent pratiquement tous les mêmes adresses. Nous aborderons ultérieurement les principales divergences qui concernent l'imprimante et le magnétophone.

Nous avons aussi deux routines en RAM d'adresse identique, envoyant à des adresses bien sûr différentes en ROM.

La première est la routine de lecture du prochain caractère à interpréter. Elle débute en # E2 ; le vecteur vers la ROM est en # F0- # F1.

L'autre est le saut à la routine d'affichage du Ready, en # 1A (26). La routine dont l'adresse est en # 1B- # 1C (27-28) a pour effet d'imprimer à l'écran le message commençant à l'adresse dont l'octet de poids faible est en A et l'octet de poids fort en T. La fin du message est signifiée par un octet nul. On peut ainsi si l'on est lassé du sempiternel Ready détourner la routine pour afficher un message saugrenu, servile ou scandaleux.

```
9 REM Ready personnalisé
10 IF DEEK(27)<>1024 THEN DOKE 0,DEEK(27)
20 DOKE 1024,#A6A5
30 DOKE 1026,#A7A4
40 DOKE 1028,#4C:DOKE 1029,DEEK(0)
50 DOKE 27,1024
60 B$=" "+CHR$(13)+CHR$(10)
70 INPUT A$
80 A$=B$+A$+B$+CHR$(0)
90 HIMEM DEEK(#A2)
```

La routine a été conçue pour pouvoir être appelée plusieurs fois. On peut à tout moment rétablir le Ready par un DOKE 27,DEEK(0). La routine machine s'écrirait LDA $A6 puis LDY $A7 et enfin JMP vers la routine en ROM. On pourrait bien sûr caser le message ailleurs, à partir de 0 par exemple, et modifier la routine en conséquence. Dans ce cas, faisons :

DOKE 1024, # A9 : DOKE 1026, # A0

pourvu que le message n'excède pas 11 caractères.

Mais il y a bien mieux à faire que cela. Ce providentiel saut nous permet d'écrire très facilement une routine VDU commune aux deux systèmes. Cette routine affiche à la position courante de l'écran le caractère contenu dans l'accumulateur et fait avancer le curseur d'une position.

```
9 REM Routine VDU commune
10 DOKE 1024,#85          'STA $00
20 DOKE 1026,#A9          'LDA #$00
30 DOKE 1028,#4CA8        'TAY
40 DOKE 1030,#1A          'JMP $1A
50 FOR C=32 TO 126:DOKE 0,C:CALL1026:NEXT
```

La ligne 50 est ici en exemple. On ne peut bien sûr appeler la routine en 1024 à partir du Basic, on doke donc C en 0 (pour être certain d'avoir un 0 en 1) et l'on appelle la routine en 1026. On a ici fait l'équivalent d'un PRINT CHR$(C);.

34 Routines mathématiques et paramétriques

Nous n'effectuerons généralement que des opérations très simples en LM, des additions et des soustractions sur 16 bits, des multiplications ou divisions impliquant des constantes. Il n'est guère envisageable de se lancer dans la conception de routines mathématiques complexes, puisque celles-ci existent déjà en ROM. Nous pourront bien sûr les appeler normalement à partir du Basic, mais nous économiserons du temps en les appelant directement à partir de notre routine LM. Il n'est pas question de faire un JSR à l'adresse de la routine COS, par exemple, sans lui transmettre le ou les paramètres voulus. Il est nécessaire pour cela d'examiner le fonctionnement de ces routines.

L'ORMOS n'effectue jamais qu'une seule opération à la fois. Les variables ou les constantes concernées par l'opération sont placées dans deux accumulateurs flottants, ACC1 et ACC2, occupant chacun 6 octets en page 0 (# D0- # D5 pour ACC1, # D8- # DD pour ACC2). La routine effectue ensuite l'opération dont le résultat est délivré en ACC1. Puis ACC1 est transféré dans la variable voulue. Les adresses # 91- # 92 contiennent l'adresse de la variable réelle lors d'une opération de transfert. Par exemple A = B/C se déroulerait ainsi :

— stockage de l'adresse de B en # 91- # 92,
— transfert de B dans ACC2,
— stockage de l'adresse de C en # 91- # 92,
— transfert de C dans ACC1,
— appel de la routine de division, résultat en ACC1,
— stockage de l'adresse de A en # 91- # 92,
— transfert de ACC1 dans A.

Toutes ces opérations sont facilement réalisables en LM, il suffit d'appeler les routines voulues. Avantage supplémentaire par rapport au Basic, nous ne devrons pas rechercher les variables que nous aurons déclarées dans un ordre déterminé. La première variable aura pour adresse DEEK(# 9C) + 2, la seconde DEEK(# 9C) + 9, etc. Nous pourrons donner au pointeur DV une valeur commode, du genre # 3FFE. La routine LM commencera par charger l'adresse # 92 avec la valeur # 40, puis il n'y aura plus à s'occuper que de l'adresse # 91, qui recevra les valeurs 0, 7, 14, etc. Cela nous permet de gérer 36 variables, ce qui sera vraisemblablement suffisant pour la plupart des applications. On peut aussi faire des opérations sur des variables indicées (# 91 recevrait alors les valeurs 0, 5, 10, etc.).

On ne peut bien sûr transférer directement une variable entière dans ACC1 ou ACC2. Il faut d'abord la convertir en variable réelle, et réciproquement.

Nous donnerons en annexe les adresses des routines mathématiques sur Oric et Atmos. A noter que les deux accumulateurs flottants se retrouvent sur la plupart des micros, avec un fonctionnement similaire. Signalons qu'il est totalement impossible d'appeler ces routines à partir du Basic, puisque l'interpréteur utilise lui aussi ces accumulateurs.

Nous avons par ailleurs les fonctions graphiques et sonores, dont nous n'allons pas donner les adresses puisque le manuel de l'Atmos s'en charge et que les équivalences ont été publiées *(V. bibliographie)*. Ces routines utilisent toutes les paramètres entiers dont les adresses sont communes aux deux systèmes (# 2E0 à # 2E8). L'intérêt de ces routines, outre le gain de temps donné par l'appel direct

en LM, réside en ce qu'une valeur incorrecte d'un paramètre ne provoquera pas l'arrêt du programme et l'affichage de l'ILLEGAL QUANTITY ERROR. Il appartiendra au programmeur de tester ou non l'adresse PARAMS qui contiendra 0 si la fonction a été exécutée, ou 1 en cas d'erreur.

Enfin, pourvu que l'on charge les paramètres avec les valeurs voulues, on peut appeler ces routines à partir du Basic.

Les routines mathématiques et paramétriques ont des adresses différentes sur les deux systèmes. Pour écrire un programme compatible les utilisant, il faudrait créer en page X une table de JMP vers les routines Oric, et en page X + 1 une table de JMP vers les routines Atmos. Le programme commencerait par tester un octet en ROM. LDA $F par exemple donne 0 sur Oric et autre chose sur Atmos. Ce BNE serait le signal qu'il faut recopier la page X + 1 en X. Le programme pourrait ensuite s'écrire simplement avec des JSR $X00, JSR $X03, etc.

35 Exemple de désassemblage

Nous espérons démontrer avec un exemple de désassemblage en ROM que le LM n'est pas si inaccessible. A partir d'un point d'entrée, on peut découvrir des pointeurs cruciaux ou des sous-routines intéressantes.

Nous avons choisi la routine RUN. Le code de RUN est 152. Pour trouver ce code sans table on entre une ligne 1 RUN et l'on fait ? PEEK(1285). On applique donc la formule *(V. 33)* qui nous fournit (sur Oric) l'adresse de la routine. Nous désassemblons donc à partir de cette adresse, en tâchant de donner en regard de chaque instruction l'équivalent BASIC :

```
C98D:  JMP $C733     =  GOTO #C733
C733:  JSR $C765     =  GOSUB #C765
C765:  CLC           :
C766:  LDA $9A       :   opérations
C768:  ADC #$FF      :  équivalentes
C76A:  STA $E9       :       à
C76C:  LDA $9B       :  AY=DEEK(#9A)-1
C76E:  ADC  #$FF     :  DOKE #E9,AY
C770:  STA $EA       :
C772:  RTS           =  RETURN
```

```
C73A:  LDA $A6        =  A=PEEK(#A6)
C73C:  LDY $A7        =  Y=PEEK(#A7)
C73E:  STA $A2        =  POKE #A2,A
C740:  STY $A3        =  POKE #A3,Y
C742:  LDA $9C        =  A=PEEK(#9C)
C744:  LDY $9D        =  Y=PEEK(#9D)
C746:  STA $9E        =  POKE #9E,A
C748:  STY $9F        =  POKE #9F,Y
C74A:  STA $A0        =  POKE #A0,A
C74C:  STY $A1        =  POKE #A1,Y
C74E:  JSR $C91F      =  GOSUB #C91F
C91F:  SEC            :
C920:  LDA $9A        :   opérations
C922:  SBC #$01       :  équivalentes
C924:  LDY $9B        :       à
C926:  BCS $C929      :  AY=DEEK(#9A)-1
C928:  DEY            :       et
C929:  STA $B0        :  DOKE #B0,AY
C92B:  STY $B1        :
C92D:  RTS            =  RETURN
```

Suivent ensuite quelques opérations de réinitialisation de la pile et de quelques pointeurs qui n'ont pas d'équivalents en Basic, et un RTS qui ramène à l'interpréteur pour l'exécution du programme.

La routine RUN exécute donc les opérations suivantes :

— le pointeur programme # E9-# EA est positionné sur le début du Basic, en principe en #500 (1280) ;
— on exécute ensuite la routine CLEAR qui remet à jour des pointeurs que nous reconnaissons au passage ;
— cette routine appelle elle-même une sous-routine RESTORE qui ramène le pointeur de DATA # B0-# B1 à 1280. Nous avons ici un exemple de la souplesse du LM et de la fantaisie avec laquelle a été conçue la ROM 1-0 : un même calcul est exécuté de deux manières différentes à quelques octets d'intervalle ;
— remise à jour de quelques pointeurs (# AC-# AD par exemple) et réinitialisation de la pile.

La gestion de la pile par le Basic est assez complexe. Un GOSUB empilera par exemple 7 éléments. Nous n'entrerons pas dans le

détail et suggérons d'éviter de sortir par une routine LM d'une boucle ou d'un sous-programme Basic.

Nous pouvons vérifier à l'aide de la formule que les routines CLEAR et RESTORE ont bien comme points d'entrée les adresses #C73A et #C91F.

Enfin nous constatons que la routine RUN n'occupe que 79 octets, ce qui démontre la concision du LM.



L'Oric est mort, vive l'Atmos ! L'Oric présentait des défauts de jeunesse, des anomalies déconcertantes qui pouvaient décourager au premier abord et inciter à condamner l'appareil. Mais il est tout à fait possible de surmonter ces quelques problèmes et de disposer ainsi d'un outil aux performances comparables à celles de l'Atmos.

Les difficultés proviennent aussi de la politique d'alcôve d'Oric Products qui tenait jalousement secrètes les adresses des variables-système et des routines de la ROM 1-0, sous l'étrange prétexte « qu'elles pouvaient changer ». Diantre ! Nous avons eu l'occasion d'observer le comportement de plusieurs Oric et nous affirmons qu'ils ne changent pas si souvent d'adresses, tels des promoteurs douteux harcelés par la vindicte populaire. Les aficionados en sont donc réduits à explorer les dédales d'une ROM touffue et brouillonnesque en éliminant à grands coups de machette les nombreux NOP et branchements inutiles.

Oui, la ROM 1-0 ressemble à un brouillon, brouillon génial peut-être, mais comportant des erreurs ou des lacunes indubitables. Ne jetons pas trop vite la pierre, la conception d'un logiciel de cet ordre présente une infinie complexité, et le debugging de la ROM 1-0 aurait retardé l'apparition de l'Oric 1 sur le marché de plusieurs mois. Mieux valait sortir cet appareil dont les possibilités restent néanmoins exceptionnelles et retravailler la conception d'une nouvelle ROM sans précipitation.

Tous ces défauts ont donc été corrigés sur la ROM 1-1, et l'Atmos dispose par ailleurs de fonctions d'entrées-sorties très améliorées. De plus, grâce à sa prétendue « définitivité », bon nombre d'adresses cruciales nous sont livrées d'emblée avec l'Atmos. Il est permis d'espérer voir divulgués tous les secrets de l'Atmos par ses concepteurs, ce qui est plus sûr que les élucubrations d'une rigueur souvent douteuse d'amateurs comme nous autres.

36 STR$

Le STR$ transforme habituellement un nombre en une chaîne de caractères. Le Basic de l'Oric place en tête de cette chaîne un CHR$(2) si le nombre est positif ou nul, ou un CHR$(45) si le nombre est négatif, ce qui est beaucoup plus normal puisque c'est le code ASCII du «-». Dans le premier cas il est impossible de récupérer la valeur numérique correspondante par la fonction inverse VAL. A noter que l'on envoie au plotter MCP-40 les paramètres des instructions graphiques sous forme de chaînes de caractères, et que les CHR$(2) font planter le plotter.

Une première solution peut être du type :

```
A$ = STR$(A):IF A>0 THEN A$ = RIGHT$(A$,LEN(A$)-1
```

Ce deviendra lassant si l'on a beaucoup de STR$ dans un programme. On peut utiliser le pointeur DC, qui contient l'adresse du premier caractère de la dernière chaîne affectée à une variable, c'est-à-dire précisément cet éventuel CHR$(2). On fera alors :

```
IF PEEK(DEEK(#A2) = 2 THEN POKE DEEK(#A2),32
```

que l'on peut transformer en sous-programme en ajoutant un RETURN. Une routine machine, n'occupant que 13 octets, présente une solution encore plus élégante puisqu'il suffira après chaque STR$ d'appeler la routine par un ! discret.

```
0 REM Correction fonction STR$
10 FOR A=1024 TO 1036
20 READ B : POKE A,B
30 NEXT
40 DATA #A0,#00        ' LDY  #$00
50 DATA #B1,#A2        ' LDA  ($A2),Y
60 DATA #C9,#2D        ' CMP  "-"
70 DATA #F0,#04        ' BEQ  RTS
80 DATA #A9,#2B        ' LDA  "+"
90 DATA #91,#A2        ' STA  ($A2),Y
100 DATA #60           ' RTS
110 DOKE #2F5,1024
120 A$=STR$(A):!:PRINTA$
```

Cette routine est équivalente à :

```
IF PEEK(DEEK(#A2)<>45 THEN POKE DEEK(#A2),43
```

43 étant le code ASCII du « + ». Il est ainsi possible de voir les résultats de cette routine sur un Atmos. On peut aussi placer plus normalement en tête de chaîne un espace en remplaçant le # 2B de la ligne 80 par un # 20. On voit mieux en hexa la nature de l'erreur, # 02 au lieu de # 20.

On aurait pu encore utiliser le pointeur # B8-# B9 :

A$ = STR$(A):IF PEEK(DEEK(DEEK(# B8) + 1)) = 2 THEN POKE DEEK(DEEK(# B8) + 1),32

37 GET

GET A$ ne fonctionne pas si l'on appuie sur la touche ', shiftée ou non (' ou "). Il faut avoir recours à la fonction KEY$ qui n'est pas vraiment équivalente, ou profiter du fait que le code ASCII de la touche enfoncée après un GET se trouve dans le premier octet du tampon clavier, en # 35 (53). On fera donc GET A$: A$ = CHR$(PEEK(53)) qui passe parfaitement dans tous les cas.

Notons qu'on peut très bien avoir un A$ = CHR$(34), ce qui correspondrait à un impossible A$ = """.

38 IF THEN ELSE

C'est une anomalie assez excentrique. Si une variable précède directement le ELSE, cela provoque un SYNTAX ERROR lorsque la condition IF est remplie. Par exemple,

4000 IF A = 0 THEN PRINT B ELSE PRINT A

ne sera pas accepté pour A = 0. On peut faire :

4000 IF A = 0 THEN PRINT B'ELSE PRINT A

ou plus lisiblement pour des non-oriciens mettre la variable en question entre parenthèses.

Cette anomalie dépend bizarrement du numéro de la ligne. On n'observe pas d'erreur pour cette même ligne en 63999 par exemple (?). Par ailleurs, lorsque c'est une variable à 2 caractères qui précède le ELSE dans une instruction entrée en mode direct, il peut arriver de curieuses choses (??).

39 HIMEM

Le HIMEM est initialisé à #9F00 sur l'Oric, c'est-à-dire vers la fin des tables de caractères en mode HIRES, ce qui signifie que ces tables seront peu à peu bouleversées en ce mode par d'éventuelles chaînes de caractères stockées par le Basic à partir du haut de la mémoire. Pour l'éviter il faut faire un HIMEM #9800 ; c'est l'adresse théorique du générateur de caractères standard, qui est initialisée sur l'Atmos.

Un GRAB va repousser le HIMEM en #BB00. Même remarque pour les tables de caractères TEXT cette fois. Il faut alors faire un HIMEM #B400, adresse théorique du générateur de caractères TEXT (en fait les adresses réelles sont #9900 en HIRES et #B500 en TEXT).

Un RELEASE va ramener le HIMEM à #9F00, même remarque.

GRAB doit être la première instruction d'un programme. On ne peut faire un HIMEM supérieur à #BAFF, mais un DOKE #A6 accompagné d'un CLEAR marche.

Il existe une anomalie assez déconcertante liée au HIMEM, au pointeur DC et à la fonction FRE(0) entre autres. Nous avouons n'y avoir rien compris, et de ce fait ne pouvons l'expliciter clairement. Il arrive qu'un programme se plantant sans raison apparente ne se plante plus (ou se plante moins) lorsqu'il est procédé à un HIMEM préalable. On voit aussi apparaître des OUT OF MEMORY ERROR lorsque l'on fait un PEEK au-delà du HIMEM, ou lorsque l'on fait un FRE(0). Une réitération de l'instruction ayant provoqué l'erreur est généralement acceptée sans discussion, mais si l'erreur s'est produite au cours d'un programme engagé dans une boucle, on ne pourra pas le relancer directement *(V. 29).*

La solution que nous employons est de parsemer un programme sujet à ces accidents de ? FRE(0) à des endroits stratégiques (hors des boucles) et d'essayer divers HIMEM. Nous ne pouvons donner d'exemple infaillible, n'en ayant pas observé de répétitif. Essayez un ?FRE(0) après un RESET, qui sait ? Et nous pouvons encore moins avancer la moindre explication. Nous nous plaisons à imaginer que la définitive dialectique digitale peut elle aussi admettre des défaillances. Gageons que l'on ne verra la fin de ces bizarres bévues binaires que le jour où l'on cessera de doper le silicium.

40 Divers : POKE, TAB, etc.

• POKE : on ne peut poker une valeur hexadécimale. Cela ne provoque pourtant aucun message d'erreur, et un Ready semble confirmer l'exécution de la commande, mais un PEEK nous montre que l'octet visé n'a pas été modifié. On ne peut donc poker directement des codes machine en hexa ; on ne peut pas non plus poker un VAL("#" + A$). Ces POKE vont tout de même peut-être quelque part. Lorsque l'on en fait en série, on peut arriver à bloquer la machine. A noter que l'adresse, elle, peut être hexa, et que l'on peut doker des valeurs hexa sans problème.

On peut utiliser ce dernier fait si l'on se soucie peu de l'octet suivant. Au lieu de faire POKE A,#EA on fera DOKE A,#EA et A + 1 sera mis à 0. Sinon il faudra se résoudre à faire la conversion, ou à faire B = #EA : POKE A,B.

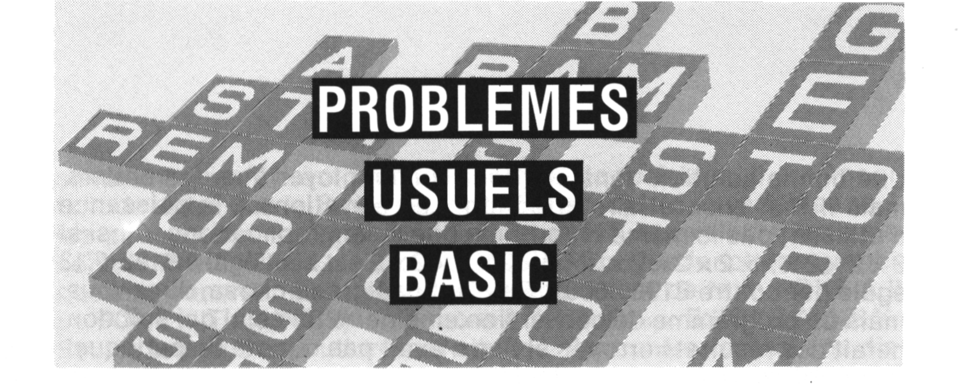
• TAB : un TAB(X) tabule en fait à X-13 et on ne peut l'employer plusieurs fois par ligne. On peut utiliser SPC(X) qui fonctionne correctement.

Par ailleurs, la tabulation par la virgule donne des choses tout à fait fantaisistes.

• DRAW : un DRAW 0,0,1 provoque de très vilaines choses à l'écran, sans arrêter le programme. Si le cas se produit, il faut l'éliminer par une ligne du genre :

```
IF X< >0 OR Y< >0 THEN DRAW X,Y,1
```

• Il existe quelques autres problèmes beaucoup moins graves, qui ne provoquent pas d'arrêt du programme. Les mélomanes n'apprécieront sans doute guère les notes programmées dans la ROM à l'usage du synthétiseur sonore. Il est possible d'y remédier en le programmant directement grâce à la routine #F535 qui stocke le contenu du registre X dans le registre du 8912 adressé par l'accumulateur. Voir la notice de ce circuit pour profiter de toutes ses possibilités, qui dépassent largement l'utilisation qu'en fait le Basic de l'ORMOS.



PROBLEMES USUELS BASIC

Il s'agit dans ce chapitre de problèmes absolument identiques pour les deux systèmes.

41 Arrondis

Ce problème est en fait commun à tous les micros. Il ne faut jamais oublier que la machine ne calcule pas comme nous : elle utilise des algorithmes qui donnent en principe de bonnes approximations, mais des approximations tout de même. Essayez par exemple ? 10.01-10, et ? 20.01-20, et ? 100000.01-100000... Bizarre, non ? Ce phénomène peut occasionner de véritables catastrophes si l'on n'y prend pas garde. Un remède consiste à n'effectuer les opérations élémentaires (+ ,—, × ,/) que sur des nombres entiers, les résultats étant alors justes. Au niveau de la représentation à l'écran, on peut avoir recours à des variables chaînes qui seront plus aisées à formatter. Voici un embryon de programme utilisant ces possibilités. Les INPUT doivent ici recevoir les données avec au maximum deux chiffres après la virgule (ou plutôt le point), positives ou négatives.

```
0 REM Comptabilité
10 M$=" ":V$=" , ":T$=" -----------
20 FOR A=9 TO 4 STEP-1:M$(A)=M$(A+1)+M$:NEXT
30 FOR A=1 TO 8 :INPUT N
40 N=N*100:T=T+N:GOSUB100:NEXT
50 A$(A)=T$:A=A+1:N=T:GOSUB100
60 FOR A=1 TO 10 :PRINTA$(A):NEXT
70 END
100 B$=STR$(N):L=LEN(B$)
110 IFLEFT$(B$,1)="-"THENS$="-"ELSES$="+"
120 A$(A)=S$+M$(L)+MID$(B$,2,L-3)+V$+RIGHT$(B$,2)
130 RETURN
```

Une bonne solution consiste donc à n'employer que des entiers, mais il faut encore faire attention. Les élévations à la puissance n'utilisent pas le même algorithme que les multiplications. Ainsi si 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 est bien égal à 8192, 2^13 égale par contre 8192.00001. Ce n'est pas grand-chose, direz-vous, mais un programme de conversion en binaire tel celui qui suit donnerait des résultats erronés si l'on n'avait pas rajouté un petit quelque chose à A pour les calculs.

```
499 REM Conversion (hexa)décimal-binaire
500 INPUT A : B=A+.5 : A$=""
510 FOR Z=15 TO 0 STEP -1
520 IF B-2^Z<0 THEN 540
530 B=B-2^Z : X$="1" : GOTO 550
540 X$="0"
550 A$=A$+X$
560 NEXT
570 PRINT A"= "HEX$(A)" = "A$
```

Essayez donc B = A avec 8192 (#2000) par exemple.

Par ailleurs, une boucle du type FOR A = 0 TO 2 x PI STEP PI/B n'arriverait pas forcément jusqu'à 2 x PI selon les valeurs de B. Il faudrait rajouter un petit quelque chose à 2 x PI pour être sûr de boucler la boucle.

Un autre exemple : COS(0) = 1, mais COS (2 x PI) = .999999998. D'une manière générale, il faudrait éviter de faire des tests d'égalité entre des variables décimales. Par exemple, au lieu de faire IF COS(X) = C il faudrait faire

IF ABS(COS(X)-C) < E

avec E dépendant de la précision à obtenir.

Remarquons que nous trouvons dans ces erreurs d'arrondis une confirmation si besoin était de l'origine du Basic de l'ORMOS. Ce sont exactement les mêmes que sur un système Microsoft.

42 Variables entières (X%)

Il est impossible d'employer les variables entières dans les boucles FOR ... NEXT, cela provoque un SYNTAX ERROR. Il est très possible de tourner le problème en employant un REPEAT ... UNTIL ou

simplement en évitant d'utiliser ce type de variables. Il est important de noter que le Basic de l'ORMOS accepte des variables réelles pour toutes les opérations nécessitant des valeurs entières, comme POKE, PEEK, AND, OR, ON, PLOT ou les fonctions graphiques ou sonores. Il est absolument équivalent de faire POKE A,B ou POKE INT(A), INT(B), ou encore A% = A:B% = B:POKE A%,B%. Seul le temps d'exécution sera allongé. Par contre, le Basic ne dispose d'aucun moyen spécifique de traiter les variables entières, et lorsqu'il en rencontre une, sa première réaction est de la convertir en variable réelle pour pouvoir la traiter, d'où perte de temps. L'intérêt principal de ces variables réside dans leur représentation facilement accessible et dans leur utilisation sous forme de tableaux, où elles n'occupent que 2 octets par élément contre 5 pour les variables réelles.

43 Interruptions

Nous préconisons d'inhiber les interruptions tant qu'un programme n'exige pas la scrutation du clavier. Mais cette interdiction a au moins deux autres conséquences en Basic :

— l'instruction WAIT n'est plus utilisable, puisqu'elle comptabilise justement ces interruptions. On peut la remplacer par des boucles FOR ... NEXT à vide ;
— on ne peut pas passer en mode HIRES sans avoir rétabli les interruptions. On peut ensuite les interdire à nouveau.

Nous avons vu que sur les deux systèmes nous pouvions interdire les interruptions au niveau du VIA par un POKE 782,64 (ou CALL #ED01 sur Oric, ou CALL #EE1A sur Atmos) et les rétablir par un POKE 782,192 (ou CALL #ECC7 sur Oric, ou CALL #EDE0 sur Atmos).

Il n'est pas nécessaire de placer cette instruction en programme. Une bonne solution est de détourner la routine du Ready en dokant en 27, #ECC7 sur Oric et #EDE0 sur Atmos. L'intérêt en est que les interruptions seront automatiquement rétablies si le programme est stoppé par une erreur quelconque, ou après un listing sur imprimante que l'on demande généralement par POKE 782,64 : LLIST pour éviter toute erreur de transmission. Bien sûr on n'a plus le Ready. Si l'on ne peut s'en passer on peut écrire une routine rétablissant le registre d'interruptions du VIA (c'est préférable car les CALL réinitialisent d'autres choses) et sauter au Ready en fin de routine. Elle s'écrirait simplement :

PHA - LDA #$C0 - STA $030E - PLA - JMP DEEK(27)
et il n'y aurait plus qu'à doker en 27 l'adresse de cette routine.

On peut aussi interdire les interruptions IRQ au niveau du 6502. A partir du Basic on pourrait faire par exemple DOKE 1024,#6078 (SEI et RTS) et DOKE 1026,#6058 (CLI et RTS). On pourra mettre un CALL 1024 en début de programme et rétablir les interruptions par un CALL 1026 ou en détournant la routine du Ready par un DOKE 27,1026, ou encore directement DOKE 26, #6058.

44 Rétablir les interruptions en cours de programme

La routine qui suit a pour objet de rétablir les interruptions en cours de programme, sans stopper celui-ci, par une pression sur le RESET. Un CTRL C ou un autre appui sur le RESET rétabli dans son rôle habituel pourront alors stopper le programme. Il n'y a plus guère de raisons maintenant de se priver du gain de temps que procure, entre autres, l'inhibition des interruptions, n'est-ce-pas ?

```
I 0400-0426
0400:   48          PHA
0401:   8A          TXA
0402:   48          PHA
0403:   98          TYA
0404:   48          PHA
0405:   A2  00      LDX #$00
0407:   A0  00      LDY #$00
0409:   C1  00      CMP ($00,X)
040B:   C8          INY
040C:   D0  FB      BNE $0409
040E:   E8          INX
040F:   D0  F6      BNE $0407
0411:   A0  00      LDY #$00
0413:   A5  02      LDA $02
0415:   91  00      STA ($00),Y
0417:   C8          INY
0418:   A5  03      LDA $03
041A:   91  00      STA ($00),Y
```

```
041C:    A9  C0        LDA #$C0
041E:    8D  0E  03    STA $030E
0421:    68            PLA
0422:    A8            TAY
0423:    68            PLA
0424:    AA            TAX
0425:    68            PLA
0426:    40            RTI
```

```
10 A=1024
20 READ A$ : IF A$="ZZ" THEN 100
30 B=VAL("#"+A$) : POKE A,B
40 A=A+1 : GOTO 20
50 DATA 48,8A,48,98,48
55 DATA A2,00,A0,00,C1,00
60 DATA C8,D0,FB,E8,D0,F6
65 DATA A0,00,A5,02,91,00
70 DATA C8,A5,03,91,00
75 DATA A9,C0,8D,0E,03
80 DATA 68,A8,68,AA,68
85 DATA 40
90 DATA ZZ
100 D=DEEK(#FFFA)+1 : POKE782,64
110 DOKE 0,D:DOKE 2,DEEK(D):DOKE D,#400
```

La routine est un peu compliquée par le fait qu'elle est valable aussi bien pour Oric que pour Atmos. Nous avons vu en effet que les points d'entrée des routines d'interruption NMI et IRQ n'étaient pas les mêmes, mais les vecteurs hardware ne sont pas si versatiles. En ligne 100 on donne donc à D la valeur 556 ou 584 et l'on inhibe les interruptions. En ligne 110 on sauvegarde cette valeur en 0-1, l'adresse de la routine en ROM en 2-3, et l'on détourne l'interruption NMI vers notre routine.

Passons à la routine proprement dite. Les cinq premières instructions sauvegardent les registres A, X et Y en les empilant, c'est une procédure classique d'interruption. Les sept instructions suivantes établissent une temporisation d'environ une seconde pour permettre de relâcher la pression sur le RESET. On exécute ici 65536 fois

une instruction absolument inutile, choisie parce qu'elle est « longue » et qu'elle ne modifie rien en RAM. Puis de # 0411 à # 041A on rétablit la routine normale d'interruptions. Ensuite on restaure le registre d'autorisations d'interruptions du VIA par l'équivalent machine d'un POKE 768,192, on récupère les registres Y, X et A et l'on sort de l'interruption par un RTI.

Une interruption IRQ sera alors immédiatement déclenchée et le programme sera stoppé par CTRL C.

Cette routine n'est pas rigide et l'on peut omettre le rétablissement de la routine normale NMI. On peut aussi réautoriser les interruptions IRQ au niveau du 6502 par un PLP - CLI - PHP inséré dans la routine. On peut aussi appeler la routine par une interruption IRQ en substituant # FFFE à # FFFA dans le programme Basic. Il est en effet possible de monter un poussoir IRQ plus commode d'accès que le RESET sans avoir à ouvrir l'appareil, ce qui permet de garder le RESET normal.

Si l'on utilise l'IRQ, il est alors possible de tester en début de routine le registre # 30E. Si les interruptions sont autorisées, on renvoie à la routine normale IRQ, sinon on exécute notre routine. On n'a pas besoin alors de rétablir le saut normal en RAM vers l'IRQ. Mais il ne faudrait utiliser le poussoir IRQ que lorsque les interruptions sont inhibées.

Recommandation : n'effectuer qu'une brève pression sur le poussoir pour ne pas déclencher une seconde interruption.

Enfin il ne faut pas maudire notre routine si le programme plante et que le RESET ne permet pas de s'en sortir. Un RESET normal serait tout aussi inefficace, à moins que la routine ait été perturbée à coups de POKE intempestifs.



Ici encore les deux systèmes divergent notablement, et l'on ne peut trouver une solution universelle commune qu'en réécrivant une routine LPRINT. Nous allons voir que ce n'est pas si difficile.

Nous sommes venu pour notre part à l'informatique parce que notre machine à écrire était en panne. Séduit par le bas prix de l'Oric, ses 48 K et son interface parallèle, nous en avons acquis un, ainsi qu'une imprimante d'assez bonne qualité, nettement plus coûteuse. Le désespoir s'abattit alors sur notre foyer : certains caractères étaient mal transmis, la machine sautait intempestivement à la ligne à la 67e colonne, ce qui avait des effets désastreux en graphisme. Le vendeur de l'imprimante nous conseilla d'acheter un Apple, le vendeur de l'Oric n'en savait pas plus que le manuel étrangement muet sur la question. Nous avons donc dû batailler pendant plusieurs semaines au PEEK et à la POKE pour obtenir un fonctionnement à peu près correct.

Le logiciel concernant l'interface parallèle a été révisé sur l'Atmos, nous ne le jugeons cependant pas aussi définitif qu'on voudrait nous le faire croire.

45 Caractéristiques communes de LPRINT et LLIST

Soulignons d'abord que LPRINT et LLIST ont un fonctionnement absolument équivalent à PRINT et LIST. LLIST n'est donc pas programmable et demande des arguments numériques.

Les adresses fondamentales sont les suivantes :

— #31 (49) définit la longueur de la ligne de l'imprimante (le nombre de colonnes),
— #30 (48) est le pointeur de la position du chariot sur la ligne. Il est incrémenté à chaque expédition d'un octet sur le port parallèle, pourvu que le poids de cet octet soit supérieur à 31.

Une instruction LPRINT « A$ » va se dérouler ainsi :

— envoi d'un caractère CHR$(X) de A$ sur le port parallèle (en commençant bien sûr par le premier),
— incrémentation du pointeur 48 si X>31,
— en cas d'égalité envoi d'un CHR$(13) et d'un CHR$(10), et réinitialisation du pointeur 48,
— retour au départ pour l'envoi du caractère suivant,
— lorsque tous les caractères ont été émis, envoi d'un CHR$(13) et d'un CHR$(10), et réinitialisation du pointeur 48.

CHR$(13) et CHR$(10) sont les codes de retour de chariot (CR) et de saut de ligne (LF) universellement admis par les imprimantes (ce sont malheureusement les seuls). Pour éviter l'envoi des CR et LF il faut terminer une instruction LPRINT par un point-virgule. Mais cela ne réinitialise pas le pointeur 48 qui nous jouera des tours lorsqu'il atteindra la valeur contenue en 49. On peut modifier cette valeur, bien sûr, mais l'octet le plus lourd du monde ne peut dépasser 255. Si c'est une valeur très confortable en mode texte, il n'en va pas de même en graphisme, les imprimantes les moins performantes offrant une résolution de 480 points par ligne. Il aurait été préférable de pouvoir se passer facilement par logiciel de ce nombre fixé de colonnes qui fait souvent double emploi avec les propres commandes de l'imprimante. A ceci près que le pointeur 48 ne serait pas réinitialisé, une instruction LPRINT A$ est donc équivalente à :

LPRINT A$; CHR$(13) ; CHR$(10) ;

Remarquons que les points-virgules ne sont nécessaires qu'en fin d'instruction ou entre deux variables réelles. Il est tout à fait possible d'écrire, sans autre inconvénient qu'une intelligibilité délicate, une instruction du type :

LPRINTA$"ORMOS"CHR$(65)BSTR$(B);

Il faut par contre écrire LPRINT A;B. Ces remarques sont valables aussi pour PRINT. Signalons à tout hasard qu'on ne peut pas écrire LPRINT sous l'éventuelle forme abrégée L? car PRINT et LPRINT sont codés par l'interpréteur sur un seul octet.

46 Aspects particuliers et remèdes personnalisés

Il est nécessaire d'aller plus au fond des choses pour apporter des solutions, et là, les deux systèmes sont quelque peu différents.

Sur Oric 1

Les adresses 48 et 49 sont ici initialisées avec les valeurs 13 et 80. Ce 13 de malheur en 48 nous fournit quelque lumière sur l'anomalie du TAB. Il n'est pas aisé de consulter ce pointeur car une instruction PRINT sans point-virgule le recharge aussi avec 13. Il s'agit probablement d'une autre bogue. Si l'on veut tabuler avec ces pointeurs il ne faut donc pas de PRINT au milieu des LPRINT. On peut envoyer jusqu'à 255 octets par ligne en faisant POKE 49,12. Mais ce n'est pas suffisant en mode graphique, sur imprimante comme sur le plotter MCP-40. Il faut alors remettre à jour le pointeur 48 soit après chaque instruction LPRINT, soit au moment opportun. On peut faire soit POKE 48,13, soit un simple PRINT. Une autre possibilité consiste à ne pas utiliser l'instruction LPRINT. On se sert alors de la routine # F57B qui envoie sur le port parallèle l'octet chargé dans l'accumulateur du 6502. Ce n'est pas une mauvaise solution car le plus souvent on expédiera les octets vers l'imprimante un par un, que ce soit en graphisme ou en traitement de texte. Il n'est pas plus long d'écrire POKE 0,X : ! que de faire LPRINT CHR$(X); , et bien sûr la routine # F57B ne se préoccupe pas du pointeur 48. Il nous faut bien entendu charger l'accumulateur avec l'octet désiré ; dans ce cas-ci on ferait tout simplement en 1024 par exemple :

```
#0400    A5 00       LDA $00
#0402    4C 7B F5    JMP $F57B
```

Soit à partir du Basic :

DOKE 1024, # A5 : DOKE 1026, # 7B4C : DOKE 1028, # F5
et DOKE # 2F5, # 400 pour le !

Reste la question des caractères mal transmis. Nous avons vu *(V. 3)* qu'il suffisait d'interdire les interruptions pour résoudre cette question. On peut le faire au niveau du VIA par un POKE 782,64 (rétablissement par POKE 782, 192), mais on peut le faire aussi au niveau du 6502 et ajouter un SEI et un CLI à notre petite routine :

```
#0400    78          SEI
#0401    A5 00       LDA $00
#0403    20 7B F5    JSR $F57B
#0406    58          CLI
#0407    60          RTS
```

On garde ainsi la main pendant des LPRINT, mais ce n'est pas possible pour LLIST, qu'il faut précéder d'un POKE 782,64 : LLIST pour obtenir un résultat impeccable. On récupère la main par un RESET ou par le détournement préalable de la routine du Ready *(V. 43)*.

Sur Atmos

Les pointeurs 48 et 49 ont un rôle un peu différent sur l'Atmos. Ils sont affectés plus généralement à la ligne d'un terminal, normalement l'écran. Ils sont donc initialisés à 2 (pour impression dans la troisième colonne de l'écran) et 40. Lors d'une instruction LPRINT, ils sont chargés avec les contenus des adresses 600 et 598 (#258 et #256) après avoir été sauvegardés en 601 et 599. Les octets 600 et 598 sont initialisés avec les valeurs 0 et 80. A la fin de l'instruction LPRINT se déroule l'opération inverse et 48 et 49 récupèrent les valeurs de 601 et 599. POS(1) est équivalent à PEEK(600). On n'a donc ici aucune possibilité d'interférence entre les PRINT et LPRINT et l'on peut ainsi utiliser avec efficacité la largeur de ligne en 598. En mode graphique toutefois, seuls seront comptabilisés les octets de poids supérieur à 31.

Pour s'affranchir des inconvénients de ce nombre de colonnes maximal de 255 (POKE 598,255) on peut recourir à la méthode vue plus haut pour l'Oric en remplaçant l'adresse 48 par 600, mais il y a une bien meilleure solution.

La routine d'envoi d'un caractère sur le port parallèle a un point d'entrée en RAM, en #23E (574), où l'on trouve un JMP $F5C1. On peut détourner cette routine pour intercaler quelques instructions annulant l'incrémentation du pointeur 48 (#30), par exemple :

```
#0400     48                  PHA
#0401     2B  équivalent officieux de LDA #$00
#0402     85  30              STA $30
#0404     68                  PLA
#0405     4C  C1   F5         JMP $F5C1
```

Soit à partir du Basic :

```
DOKE 1024, #2B48 : DOKE 1026, #3085 :
DOKE 1028, #4C68 : DOKE 1030, #F5C1
```

et DOKE 575,1024 pour détourner la routine. On peut ainsi utiliser LPRINT sans se préoccuper de la largeur de ligne. On ne peut faire un DEC $30 seul qui décrémenterait aussi 48 pour les codes CR et LF.

La routine #F5C1 comporte un SEI inhibant les interruptions, mais celles-ci sont rétablies avant le test sur la réception du ACK de l'imprimante. Il semblerait qu'il y ait là un risque d'erreur, dans le cas où une interruption demandant la scrutation du clavier interviendrait avant la réception du ACK. Le risque est sans doute faible, mais nous conseillons d'inhiber les interruptions comme pour l'Oric, par l'une ou l'autre des méthodes vues plus haut.

47 Une routine machine LPRINT universelle

L'instruction LPRINT est fondamentale (surtout pour ceux qui possèdent une imprimante !), il serait donc souhaitable de trouver une solution pouvant tourner sur les deux systèmes. La routine que nous présentons envoie sur le port parallèle le contenu de l'accumulateur, tout comme les routines # F57B de l'Oric et # F5C1 de l'Atmos. Elle a cependant les caractéristiques particulières suivantes :

— elle teste le ACK de l'imprimante avant d'expédier la donnée : il n'y a en conséquence aucune perte de temps, l'ORMOS retournant au turbin pendant que l'imprimante digère la donnée,
— ceci impose que les interruptions soient interdites,
— telle qu'elle est présentée, la routine stocke aussi les données au-dessus du HIMEM : il est ainsi possible, pour un programme consacrant beaucoup de temps à la préparation des données, d'appeler, l'impression terminée, un sous-programme permettant une seconde impression bien plus rapide.

Voici tout de suite la routine principale, expédiant le contenu de l'accumulateur vers l'imprimante :

```
Routine Principale expédiant le contenu
de l'Accumulateur vers l'Imprimante
    I 0400 - 0422
0400:   AA              TAX
0401:   AD  0D  03      LDA $030D
0404:   29  02          AND #$02
0406:   F0  F9          BEQ $0401
0408:   8E  01  03      STX $0301
040B:   AD  00  03      LDA $0300
040E:   A8              TAY
040F:   29  EF          AND #$EF
0411:   8D  00  03      STA $0300
0414:   8C  00  03      STY $0300
0417:   8A              TXA
0418:   A0  00          LDY #$00
041A:   91  00          STA ($00),Y
041C:   E6  00          INC $00
041E:   D0  02          BNE $0422
0420:   E6  01          INC $01
0422:   60              RTS
```

Et voici un exemple en Basic d'acquisition de donnée (variable entière → accumulateur).

```
Exemple d'acquisition de donnée
Variable entière --> Accumulateur
    I 0423 - 0427
0423:    A0  01        LDY #$01
0425:    B1  B8        LDA ($B8),Y
0427:    4C  00  04    JMP $0400
```

```
99 REM Exemple d'application
100 HIMEM 10000:DOKE 0,10000
110 INPUT A$:A$=A$+CHR$(13)+CHR$(10)
120 POKE 782,64
130 FOR A=1 TO LEN(A$)
140 X%=ASC(MID$(A$,A,1)):CALL #423
150 NEXT:POKE 782,192:STOP
19999 REM Réimpression des données
20000 B=DEEK(0):HM=DEEK(#A6)
20010 DOKE 0,HM:POKE 782,64
20020 FOR A=HM TO B-1
20030 X%=PEEK(A):CALL #423
20040 NEXT:POKE 782,192:STOP
```

La fonction de stockage n'est pas obligatoire et l'on peut très bien stopper la routine en #0417 par un RTS. Remarquons qu'il serait possible en Basic d'obtenir une fonction analogue en expédiant toutes les données X ainsi :

A$ = CHR$(X) : LPRINT A$;

On suppose ici résolus les problèmes du pointeur 48, de l'envoi des CR et LF. Pourvu que A$ soit la seule variable chaîne employée dans le programme, on retrouvera toutes les valeurs successives de X à partir du HIMEM, vers le bas cette fois. On pourra réimprimer le document en faisant :

A = DEEK(# A6) : B = DEEK(# A2)
REPEAT : A = A-1 : LPRINT CHR$(PEEK(A)); : UNTIL A = B

Pour les enragés de vitesse, on peut utiliser la routine en # 401 (1025) pourvu que ce soit le registre X qui soit chargé avec la donnée. On

peut encore gagner quelques microsecondes avec le POKE #30C,220 *(V. 12).*

On peut bien sûr utiliser la routine avec l'instruction LPRINT de l'Atmos en rentrant un DOKE 575, 1024, toujours à condition que les interruptions soient inhibées.

Si l'imprimante dispose de l'AUTO LINE-FEED, on peut éviter d'envoyer le LF après le CR, ou s'en servir pour imprimer avec double interligne.

Un dernier détail : il arrive que la mise sous tension de l'imprimante provoque un plantage non rissettable de l'ORMOS (j'y pense car ça vient de m'arriver, et la dernière page a été récupérée par le SDEC 002). Il vaut donc mieux brancher l'imprimante avant de lancer le programme, mais pas avant de brancher l'ORMOS ou de charger un programme, ces opérations provoquant l'expédition d'un caractère parasite.

48 LIST et LLIST programmables

Il est bien difficile de trouver une solution simple pour programmer LLIST. Nous vous livrons cependant cette routine...

```
8 REM 4 Lignes en Exemple
9 REM pour un Listing Intégral
10 LL=DEEK(#9A)
20 REPEAT
30 N=DEEK(LL+2):GOSUB 1000:LL=DEEK(LL)
40 UNTIL DEEK(LL)=0:STOP

999 REM LIST (ou LLIST) programmable
1000 L=DEEK(#9A)
1010 REPEAT
1020 NL=DEEK(L+2):P=L:L=DEEK(L)
1030 UNTIL N=NL
1040 PRINT N;
1050 FOR A=P+4 TO L-2
1060 C=PEEK(A)
1070 IF C<127 THEN PRINT CHR$(C);:GOTO 1160
1080 C=C-127:D=#C0E9:E=D
1090 REPEAT
```

```
1100 IF PEEK(D)>128 THEN C=C-1:E=D
1110 D=D+1
1120 UNTIL C=0
1130 REPEAT
1140 E=E+1:PRINT CHR$(PEEK(E));
1150 UNTIL PEEK(E)>128
1160 NEXT:PRINT:RETURN
```

La routine principale liste sur l'écran, ou sur l'imprimante si l'on remplace les PRINT par des LPRINT, la ligne de numéro N. La clef est l'utilisation de la table des mots clefs dont l'adresse de début est identique sur les deux systèmes (#C0EA).

Les lignes 10 à 40 donnent un exemple d'utilisation de la routine, exemple un rien aberrant puisqu'il va falloir cinq minutes pour lister ces quelques malheureuses lignes, mais c'est programmable, et nous ne doutons pas que vous saurez trouver d'intéressantes applications de cette routine.



Il est extrêmement important de sauvegarder non seulement un programme, mais aussi les données variables qui lui sont associées. Celles-ci sont souvent le résultat de longues heures de travail aussi bien de l'ordinateur que du programmeur.

Mis à par les bogues revues et corrigées sur diverses instructions, c'est essentiellement au niveau de l'utilisation du magnétophone que divergent les deux systèmes. Nous allons donc dans ce chapitre dissocier l'Oric de l'Atmos. Nous invitons toutefois les possesseurs d'Atmos à ne pas sauter les quelques pages consacrées à son vénérable prédécesseur.

49 Instructions CSAVE et CLOAD de l'Oric

Première constatation qui s'impose : il n'y a aucune commande Basic qui permette de sauvegarder les variables. Il existe néanmoins la possibilité de sauvegarder des zones de mémoire par l'ordre CSAVE, A XXXX, E YYYY.

Deuxième constation : lorsque l'on relit de tels blocs de mémoire et que l'on tente de relancer le programme, on obtient bien souvent des OUT OF MEMORY ERROR aussi constants que consternants.

Troisième constatation : si l'ordre CSAVE est utilisable en mode programmé, l'ordre CLOAD interrompt le programme et envoie un Ready, excepté si la sauvegarde avait été faite avec l'option AUTO.

Ces deux derniers problèmes proviennent d'une bogue du Basic 1.0. Sur l'Atmos l'instruction CSAVE A XX, E YY positionne un indicateur. A la relecture, la présence de cet indicateur informe l'interpréteur qu'il ne charge pas du Basic.

Cet indicateur existe bel et bien sur l'Oric, mais il n'est pas positionné par l'instruction CSAVE. Un ordre CSAVE"" est en fait absolument équivalent à un ordre CSAVE"", A DEEK(#9A), E DEEK(#9C).

Un ordre CLOAD"" va charger en mémoire la zone sauvegardée aux adresses correspondantes, puis il va charger le pointeur DV avec la valeur de la dernière adresse lue. Les pointeurs DT et FT gardent leur ancienne valeur. Si l'on a chargé une zone protégée par le HIMEM, le OUT OF MEMORY ERROR est bien compréhensible. Et si l'on a chargé des variables Basic, le réajustement du pointeur DV les rend inutilisables. Il est toujours possible de bricoler en mode direct pour récupérer ces variables et réajuster les pointeurs *(V. 27)*, mais on ne peut le faire en mode programmé, puisque CLOAD arrête le programme. La clef du problème consiste à appeler directement d'autres points d'entrée des routines intéressant l'interface cassette et à positionner soi-même les pointeurs et indicateurs concernés.

Voici les adresses utiles :

— A , adresses #5F-#60 (95-96) : pointeur contenant l'adresse du premier octet sauvegardé ou lu selon l'opération effectuée,
— E , adresses #61-#62 (97-98) : pointe sur le dernier octet lu ou à sauvegarder,
— AUTO, adresse #63 (99) : si cet indicateur contient une autre valeur que 0, on est en mode AUTORUN,
— BM, adresse #64 (100) : si cet octet est à 0, la sauvegarde est supposée être du Basic, sinon du code machine,
— FS, adresse #67 (103) : si cet octet est à 0, sauvegarde rapide, sinon lente,
— TP, adresses à partir de #35 (53) : c'est à partir de cet octet qu'est stocké en ASCII le nom de la sauvegarde ; sa fin est notifiée par un 0. Toutes les autres valeurs sont permises. Si l'on a fait un CSAVE "" on aura un 0 en #35,
— #E6CA : cette routine initialise le VIA pour l'interface cassette, elle inhibe les interruptions et démarre le magnétophone,
— #E804 : rétablit les registres du VIA,
— #E57B : routine d'écriture,
— #E4A8 : routine de lecture.

Lorsque l'on fait un CSAVE "NOM" avec des options quelconques, l'interpréteur va positionner les pointeurs et indicateurs selon les options désirées. Par défaut on aura donc A à DB, E à DV, AUTO, BM, FS et TP à 0. Puis l'Oric enverra quelques octets d'en-tête, le nom de la sauvegarde, le contenu de ces différents pointeurs et enfin la sauvegarde elle-même. Lors de la lecture ce nom sera comparé à celui qui se trouvera dans le tampon. S'il y a concordance les octets lus seront chargés en mémoire, à commencer par les pointeurs. Si la lecture se produit correctement les octets de E seront recopiés en DV, et les pointeurs DT et FT ajustés à ce niveau. A noter

que le pointeur DB n'est pas repositionné au niveau de A. Au lieu de modifier les deux pointeurs en Basic et de n'en modifier aucun en machine, l'Oric a choisi une curieuse solution moyenne qui ne satisfait personne. Il faut donc rentrer « à la main » la bonne valeur du DB d'un programme Basic ne débutant pas en 1281.

50 Sauvegarde d'une zone mémoire sur ORIC 1

Nous venons de voir que les instructions CSAVE et CLOAD étaient d'un usage délicat et qu'il était impossible de repositionner en programme les pointeurs affectés par un CLOAD puisque celui-ci stoppe le programme (sauf en AUTO).

Signalons tout de même un cas intéressant : si l'on veut sauvegarder avec un programme Basic quelques routines machine en page 4, il suffit de faire un CSAVE "NOM", A 1024.

Dans les autres cas il faut faire appel à deux routines d'écriture et lecture de ce genre :

```
5000 REM Sauvegarde zone mémoire ORIC 1
5010 INPUT "Adresse premier octet";A
5020 INPUT "Adresse dernier octet";E
5030 INPUT "Nom de la sauvegarde";K$
5040 INPUT "Rapide ou Lente (R/L)";R$
5050 IF R$="R" THEN POKE #67,0:GOTO 5070
5060 IF R$="L" THEN POKE #67,1 ELSE GOTO 5040
5070 POKE #63,0:POKE #64,1
5080 DOKE #5F,A:DOKE #61,E
5090 K$=K$+CHR$(0)
5100 FOR A=1 TO LEN(K$)
5110 B=ASC(MID$(K$,A)):POKE A+52,B:NEXT
5120 CALL #E6CA:CALL #E57B:CALL #E804
5130 RETURN
5500 REM Chargement zone mémoire ORIC-1
5510 INPUT "Nom de la sauvegarde";K$
5520 INPUT "Rapide ou Lente (R/L)";R$
5530 IF R$="R" THEN POKE #67,0:GOTO 5550
5540 IF R$="L" THEN POKE #67,1 ELSE GOTO 5520
5550 K$=K$+CHR$(0)
```

```
5560 FOR A=1 TO LEN(K$)
5570 B=ASC(MID$(K$,A)):POKE A+52,B:NEXT
5580 CALL #E6CA:CALL #E4A8:CALL #E804
5590 RETURN
```

Les INPUT ne sont bien entendus présentés ici qu'à titre d'exemple. A noter que ces opérations sont impossibles à exécuter en mode direct, puisque les pointeurs et indicateurs divers se trouvent dans le tampon clavier.

Un cas particulier se pose, la sauvegarde de l'écran HIRES pendant laquelle apparaissent, sur la ligne correspondant à la ligne TEXT de STATUS, les messages divers, aussi bien en lecture qu'en écriture. La solution consiste à recopier les 8000 octets de l'écran HIRES dans une zone tranquille, et à effectuer la sauvegarde sur cette zone. Il faudra ensuite à la lecture faire le transfert inverse après un HIRES. Ces opérations sont presque instantanées en langage machine *(V. 60)*, mais on peut les réaliser en Basic si l'on n'est pas trop pressé, comme suit par exemple :

```
HIMEM #7000 'en début de programme
B = #3000
FOR A = #A000 TO #BF3E STEP 2
DOKE A-B, DEEK(A)
NEXT
```

On peut alors effectuer la sauvegarde avec A = #7000 et E = #8F. On récupérera un écran après une lecture par :

```
HIRES : B = #3000
FOR A = #A000 TO #BF3E STEP 2
DOKE A, DEEK(A-B)
NEXT
```

La valeur de B est bien entendu optionnelle.

51 Sauvegarde des variables

sur Oric 1

Examinons tout de suite ces deux sous-programmes qui permettent de sauvegarder à la fois les variables non indicées, les tableaux et les chaînes de caractères.

A noter que les lignes 6020 à 6035 et 7020 à 7035 sont en fait inutiles dans ce cas précis, car après un INPUT K$ se trouvera dans le

tampon clavier exactement sous la forme désirée. Mais ces sous-programmes peuvent être modifiés et ces lignes seront alors nécessaires si K$ ne provient plus d'un INPUT.

```
6000 REM Sauvegarde des données ORIC 1
6010 INPUT "Nom de la sauvegarde";K$
6020 K$=K$+CHR$(0)
6030 FOR A=1 TO LEN(K$)
6035 B=ASC(MID$(K$,A)):POKE A+52,B:NEXT
6040 PRINT FRE(0)
6050 POKE #63,0:POKE #64,1:POKE #67,0
6060 DOKE #5F,DEEK(#9C):DOKE #61,DEEK(#9E)
6070 CALL #E6CA:CALL #E57B
6080 DOKE #5F,DEEK(#9E):DOKE #61,DEEK(#A0)
6090 CALL #E57B
6100 DOKE #5F,DEEK(#A2):DOKE #61,DEEK(#A6)
6110 CALL #E57B:CALL #E804
6120 RETURN
7000 REM Lecture des données ORIC 1
7010 INPUT "Quel fichier voulez-vous lire";K$
7020 K$=K$+CHR$(0)
7030 FOR A=1 TO LEN(K$)
7035 B=ASC(MID$(K$,A)):POKE A+52,B:NEXT
7040 CALL #E6CA:CALL #E4A8
7050 DOKE #9E,DEEK(#61)
7060 CALL #E4A8
7070 DOKE #A0,DEEK(#61)
7080 CALL #E4A8:CALL #E804
7090 DOKE #A2,DEEK(#5F)
7100 RETURN
```

Peu de commentaires sont nécessaires puisqu'il ne s'agit que d'une variante des sous-programmes vus dans la section précédente. Le truc consiste à sauver en trois fois les zones variables, tableaux et chaînes. On retourne ensuite au programme appelant. A la relecture on charge en trois fois également les trois zones en repositionnant les pointeurs DT, FT et DC.

Ces deux sous-programmes sont bien adaptés à la sauvegarde des données d'un programme unique invariant. On peut néanmoins

transmettre ces données à un autre programme pourvu que celui-ci ne soit pas plus long. Il suffit alors de positionner le pointeur DC en 7050 comme suit :

7050 DOKE #9C, DEEK(#5F) : DOKE #9E, DEEK(#61)

On peut en faire de même pour le HIMEM ainsi :

7090 DOKE #A2, DEEK(#5F) : DOKE #A6, DEEK(#61)

Dans le cas où l'on voudrait transmettre des données à un programme plus long que le programme initial, il faudrait positionner au départ des différents programmes le pointeur DV avec une même valeur confortable *(V. 30)*. Il est également souhaitable que ces programmes aient le même HIMEM.

Nous jugeons cette méthode assez complète pour ne plus avoir à chercher mieux. Citons néanmoins quelques autres solutions dignes d'un certain intérêt, utilisées avant cette découverte définitive. Une méthode consistait à placer la zone variables à l'intérieur du programme, en dokant en page 4 les contenus des pointeurs à restaurer *(V. 25, 29)*. Il fallait bien sûr sauvegarder tout le programme à chaque opération. Une autre méthode, parce que nous avions des difficultés à sauvegarder les tableaux, recourait à un curieux artifice, grâce à deux lignes en début de programme :

20 AA$ = A$: AA = A : AA% = A% : RETURN
30 A$ = AA$: A = AA : A% = AA% : RETURN

Il fallait déterminer les adresses des AA et y poker les ASCII correspondant à des caractères valides de noms de variables avant d'appeler la ligne 20 ou 30 selon que l'on désirait affecter une variable ou lire sa valeur. La méthode permet d'obtenir trois pseudo-tableaux en zone des variables simples, de dimensions 26 × 36. Le problème est qu'il n'est alors possible de donner aux autres variables que des noms d'une seule lettre. Nous en étions réduits à employer des variables indicées à la place des variables d'usage courant.

Mais nous n'allons pas passer en revue toutes les méthodes baroques ou loufoques, ingénieuses ou infructueuses, que nous avons développées pour tâcher de préserver ces mortelles données d'un irrémédiable effacement. Nous nous bornerons à constater que notre méthode est bien plus simple que celle diffusée par Oric qui nécessite de charger une bonne tripotée de barbare code hexa (nous avouons n'avoir jamais réussi à le faire tourner), et qu'elle est d'un usage bien plus général.

sur Atmos

On observe ici un très réel progrès par rapport à l'Oric. Les options J et V sont extrêmement utiles. Un point noir pour le Errors found qui empêche bien souvent l'AUTORUN.

Nous avons aussi les fonctions STORE et RECALL, que nous trouvons un peu faiblardes, d'un emploi difficile et contraignant. Précisons que ce n'est pas l'interpréteur qui stocke les tableaux de la même manière quelle que soit la nature des éléments, c'est la routine elle-même qui ne prend en compte que le premier caractère. En bref, ces fonctions valent mieux que rien, elles seront bien suffisantes si l'on n'a qu'un seul tableau à sauvegarder, mais dans bien d'autres cas nous préférerons faire appel à une routine similaire à celle vue pour l'Oric, qui permet de sauvegarder toutes les variables et de les recharger sans avoir à redimensionner les tableaux. Cette routine s'écrit bien plus simplement sur Atmos puisqu'on peut employer les instructions CSAVE et CLOAD qui fonctionnent bien (CLOAD programmable). Les adresses des pointeurs ont changé :

— A , l'adresse du premier octet de la sauvegarde, se situe en # 2A9-# 2AA (681-682),
— E , l'adresse du dernier octet de la sauvegarde, se trouve en # 2AB-# 2AC (683-684),
— les indicateurs AUTO, BM et FS occupent les adresses # 2AD, # 2AE et # 24D (685, 686 et 589), mais nous n'aurons pas besoin de les utiliser puisqu'ils sont correctement positionnés par le CSAVE.

Voici donc les deux sous-programmes :

```
6000 REM Sauvegarde des données ATMOS
6010 INPUT "Nom de la sauvegarde";K$
6020 PRINT FRE(0)
6030 CSAVE K$, A DEEK(#9C), E DEEK(#9E)
6040 CSAVE K$, A DEEK(#9E), E DEEK(#A0)
6050 CSAVE K$, A DEEK(#A2), E DEEK(#A6)
6060 RETURN
7000 REM Lecture des données ATMOS
7010 INPUT "Quel fichier voulez-vous lire";K$
7020 CLOAD K$ : DOKE #9E,DEEK(#2AB)
7030 CLOAD K$ : DOKE #A0,DEEK(#2AB)
7040 CLOAD K$ : DOKE #A2,DEEK(#2A9)
7050 RETURN
```

Voir le paragraphe précédent pour le fonctionnement et les applications. Rappelons qu'il est possible de transmettre des jeux de variables entre différents programmes pourvu que ceux-ci aient les mêmes DV et HM. On peut calculer une valeur confortable du DV tenant compte d'ajouts éventuels aux programmes. Ces programmes débuteront tous en principe par une ligne de ce type avec les valeurs de DV et HM bien sûr optionnelles :

10 DOKE #9C, 10000 : HIMEM 30000

Notons qu'il est ainsi possible de charger un tableau dont on ne connaît pas les dimensions. On écrit facilement une routine qui lit, une fois le tableau chargé, le nombre de ses dimensions en DT + 4 et le nombre d'éléments dans chaque dimension dans les octets suivants *(V. 24)*.

On peut aussi écrire des programmes extrêmement longs dont on chargera les différentes étapes en mode AUTORUN. Il ne sera pas nécessaire de recharger les blocs de variables dont on aura pu doker les pointeurs dans une zone tranquille à l'abri du Basic pour les restaurer au début de l'étape suivante.

OPTIMISATION DES PROGRAMMES

Il ne s'agit pas ici d'apprendre à programmer, mais de livrer quelques trucs permettant d'améliorer un programme existant. On suppose bien sûr que ce programme a déjà une structure logique optimale, et pour le moins qu'il tourne.

Cette optimisation va porter sur deux points fondamentaux, la rapidité d'exécution et la place occupée en mémoire. Vu que les exigences sur ces deux points ne sont pas forcément les mêmes, nous présentons les possibilités en trois catégories.

L'ORMOS est un système lent ; plus que sur un autre système il sera nécessaire d'optimiser le logiciel pour en obtenir la rapidité d'exécution maximale. Il n'est guère possible de donner ne serait-ce qu'un ordre de grandeur du temps que l'on peut gagner en optimisant. On arrive assez facilement à diminuer de moitié le temps d'exécution, mais un facteur de l'ordre de 10 ou plus n'est pas exceptionnel, selon les programmes. Au cas où l'on arriverait pas à un résultat satisfaisant, il faudrait avoir recours au langage machine.

Bien sûr l'ORMOS a 48 K de RAM largement suffisants pour la plupart des applications. Mais des programmes de fichiers, de jeux graphiques ou de graphisme sur imprimante occupent beaucoup de place. Dans ce dernier cas par exemple, on peut imprimer en simple densité 576 points par ligne en format A4. Un carré comportera dans ces conditions 331 776 points, ce qui représente 41 472 octets. De mémoire d'ORMOS on commence à s'inquiéter.

Un programme complètement optimisé devient très facilement aussi totalement incompréhensible, même pour son propre créateur. Il est donc conseillé de ne procéder à cette opération qu'en dernier lieu, et de garder sous la main un listing et des sauvegardes du programme en clair, aux fins de modifications ultérieures.

Enfin, la plupart des trucs proposés sont valables pour d'autres systèmes.

52 Gagner temps et place en RAM

Alors pourquoi s'en priver ? Il s'agit essentiellement, pour ne conserver que ce dont l'interpréteur a besoin, de compiler le texte Basic, de supprimer tout ce qui est inutile :

les espaces

Ils ne sont pas pris en compte par l'interpréteur, excepté ceux qui se trouvent à l'intérieur d'une chaîne de caractères.

les REM

Sans commentaires...

les ; dans PRINT et LPRINT

Ils sont inutiles excepté entre deux variables ou en fin d'instruction.

les '' en fin de ligne

L'interpréteur se verra signifier la fin de la chaîne par le zéro de fin de ligne. Exemples :

```
PRINTCHR$(A)STR$(B)A$D$A;BLEFT$(A$,X)''ABC
CLOAD''
```

les parenthèses

Ici il faut faire très attention aux priorités. En ce qui concerne les opérations arithmétiques, l'ORMOS calcule d'abord ^, puis × et /, puis + et —. Pour les tests et opérateurs logiques, les tests (=,<,>,=<,=> et <>) sont d'abord effectués, puis NOT, puis AND, et enfin OR. Par ailleurs les parenthèses sont parfois nécessaires pour isoler une variable d'une instruction. On doit faire IF((T)ORA)=PATHEN... pour que l'interpréteur n'isole pas d'abord le mot clef TO.

les INT inutiles

Toutes les fonctions de l'ORMOS demandant des valeurs entières acceptent en fait des valeurs décimales qui seront converties exac-

tement de la même façon que par la fonction INT (ou que par un passage en variable entière).

n'employer que des noms de variables courts

Donc une lettre (plus $ ou % selon les cas), tant que c'est possible, et deux caractères au maximum. Attention aux mots réservés du Basic.

écrire un nombre maximal d'instructions par ligne

Ou écrire un nombre minimal de lignes... Nous avons vu *(V. 31)* qu'il était possible de ne plus être limité par les 80 octets du tampon clavier. On ne peut bien sûr écrire une nouvelle instruction après un GOTO. Il ne faut jamais hésiter à employer des ELSE dans les instructions IF THEN. En bref, une ligne de moins, c'est 4 octets d'économisés et quelques microsecondes de gagnées.

ne pas donner le nom des variables après NEXT

L'interpréteur n'en a nul besoin, les valeurs concernant la boucle en cours (adresse de la variable, valeur d'arrivée, pas) étant stockées dans la pile du 6502. Sinon il perdra du temps à chercher la variable en question sans autre bénéfice qu'un risque d'erreur. Et qui ne s'est jamais trompé en imbriquant un certain nombres de boucles, alors qu'il suffit de compter les FOR et de terminer par une ligne de NEXT:NEXT:NEXT en nombre idoine ?

53 Accroître la rapidité d'exécution

Les méthodes vues dans la section précédente restent valables. On peut toutefois réintégrer les REM ou les commentaires qui sont tout de même bien utiles si l'on veut y comprendre quelque chose. On s'arrangera pour les placer dans des lignes qui ne seront jamais exécutées.

Ensuite il faut agir avec méthode et déterminer quelles sont les parties « longues » du programme. S'il se présente sous une forme du type :

```
FOR A=1 TO 10 : faites ceci
FOR B=1 TO 10 : faites cela
FOR C=1 TO 10 : faites autre chose
NEXT C,B,A
```

il faudra en principe se concentrer sur les instructions comprises dans la boucle C, car elles seront exécutées 1000 fois, contre 100 et 10 pour B et A. On pourra bien sûr optimiser tout le programme, mais les gains les plus significatifs se feront à l'intérieur des boucles ou sous-programmes le plus souvent exécutés. Et si les résultats n'étaient pas satisfaisants, c'est encore à l'intérieur de ces lignes qu'il faudrait rechercher quelles instructions pourraient être facilement remplacées par une routine machine.

remplacer toutes les constantes par des variables

Ceci se fera en début de programme. On peut aussi faire des GOTO et des GOSUB alphanumériques, ce qui ne peut d'ailleurs que clarifier la programmation et permettre une renumérotation plus aisée. Exemple :

```
10 P1=1000:P2=1500:K=337
570 IFA>KTHENP1ELSEP2
```

déclarer en début de programme les variables selon leur fréquence d'utilisation

Les premières variables affectées seront en effet plus vite débusquées par l'interpréteur, et le programmeur connaîtra ainsi leur emplacement.

Par ailleurs il faudra veiller à réserver aux variables les plus employées les noms d'une seule lettre.

Remarquons que le fait d'écrire les constantes sous la forme de caractères uniques permet aussi de gagner de la place dans la ligne et d'y caser plus d'instructions.

les chaînes de caractères

Ces deux derniers points s'appliquent aussi aux chaînes. Il arrive que l'on ait à appeler fréquemment certains CHR$() ou groupes de CHR$(), notamment pour les codes de contrôle de l'imprimante ou la gestion de l'écran. On a tout avantage à tous les niveaux à les remplacer par des variables. On les déclarera aussi selon leur ordre d'importance.

éviter d'employer les variables entières

Répétons donc que l'interpréteur les transformera avant toute autre opération en variables réelles.

créer des tables de valeurs

Si certains calculs impliquant des valeurs fixes doivent se répéter, on peut avoir intérêt à les effectuer une fois pour toutes en début de programme, et à stocker les résultats sous forme de tableaux. Exemple :

```
DIM X(360), Y(360)
FOR A=0 TO 360
B=A×PI/180:X(A)=R×COS(B):Y(A)=R×SIN(B)
NEXT
```

On a ainsi les coordonnées directement accessibles des points du cercle de rayon R, par pas d'un degré.

les tests

Si l'on doit comparer une variable à plusieurs valeurs fixes, point n'est besoin d'effectuer les tests. Au lieu de

```
IF A=10 THEN GOTO 110
IF A=20 THEN GOTO 120, etc.
```

on peut bien sûr faire GOTO 100+A. Lorsque ce n'est pas possible et que les tests sont néanmoins nombreux, il faut faire appel à des tests préliminaires pour diminuer autant que possible le nombre de tests successifs à effectuer sur une variable.

Si l'on utilise une variable indicatrice ne pouvant prendre que deux états, on a intérêt à prendre 0 pour l'un des états. On peut ainsi faire IF A THEN..., le test étant considéré par l'interpréteur comme réussi si A< >0. On peut aussi utiliser dans une expression un test entre parenthèses qui vaut —1 ou 0. Exemple :

```
GOTO 100 —10×(C<A) — 150×(C=A/ — 600*(C>A)
```

A noter que les tests entre fonctions chaînes se font plus vite que les tests entre valeurs numériques. Ainsi, (B$=CHR$(13)) se fera plus vite que (ASC(B$)=13). Il y aurait beaucoup plus de choses à dire sur les tests. Précisons encore que la compilation de lignes Basic permet dans certains cas de mettre énormément d'instructions dans une seule ligne IF THEN ELSE et donc d'éviter éventuellement des branchements *(V. 31).*

les GOSUB

En principe il faudrait les éviter et répéter quelques lignes plutôt que d'effectuer deux instructions supplémentaires (l'éditeur de l'OR-

MOS n'est pas des plus satisfaisants, mais il permet néanmoins de recopier facilement des groupes de lignes). Si l'on doit tout de même en employer, il est recommandé de placer les sous-programmes en début de programme, afin de ne pas avoir à parcourir tout le texte Basic pour dénicher la ligne appelée. C'est bien sûr aussi vrai pour les GOTO, mais bien moins évident à mettre en œuvre.

les interruptions

On peut soit les interdire totalement (POKE 782,64), soit diminuer leur fréquence (POKE 775,255 au maximum) pour bénéficier encore du CTRL C.

Il y aurait certainement bien d'autres choses à dire et cette liste n'est pas exhaustive.

54 Gagner de la place en RAM

Dans le cas le plus général nous ne toucherons plus guère au programme Basic qui occupera vraisemblablement une place minime au sein de la mémoire. Il nous faudra donc gagner des octets en tâchant de diminuer la place occupée par les variables, et principalement par les tableaux de variables.

les variables entières

C'est la solution Basic la plus évidente. Si A est compris entre —32768 et + 32767 on aura tout avantage à faire DIM A%(9,9), occupant 209 octets, plutôt que DIM A(9,9), occupant 509 octets.

Si A n'est pas compris entre ces valeurs, mais qu'une précision de l'ordre de 1/65 536e nous suffit, on aura tout intérêt encore à transformer A de manière à pouvoir employer les tableaux d'entiers. Par exemple, si A est compris entre 0 et # FFFF, on fera :

A%(X,Y) = (65536—A) × (A > # 7FFF) — A × (A < × 8000)

et pour relire le fichier :

B = A%(X,Y):A = —(65536 + B) × (B < 0) — B × (B > = 0)

Si A est compris entre —1 et + 1 (cosinus par exemple), on pourra faire, avec K = # 7FFF (32767) :

A%(X,Y) = A × K

Dans l'autre sens :

A = A%(X,Y)/K

Dans bien des cas la précision obtenue sera tout à fait suffisante, et le gain en octets appréciable.

les variables chaînes de caractères

On peut faire véhiculer par des variables chaînes quantité d'informations, à concurrence de 255 octets. Par exemple, si l'on a une variable A inférieure à 256 associée à une chaîne A$, point ne sera besoin de faire DIM A$(X,Y), A%(X,Y). Il suffira d'inclure A dans la chaîne en faisant :

A$(X,Y) = CHR$(A) + A$(X,Y)

et de récupérer les deux variables en faisant par exemple :

A = ASC(A$(X,Y) et A$(X,Y) = MID$(A$(X,Y),2)

On peut ainsi transmettre plusieurs paramètres associés à une chaîne, mais également transformer directement une série de variables quelconques en une seule chaîne, selon toutes les codifications imaginables.

profiter au maximum de la mémoire

S'il manque au bout du compte quelques K, on peut récupérer les zones correspondant aux tables de caractères et à l'écran TEXT. Un GRAB:HIMEM # C000 provoque un OUT OF MEMORY ERROR, mais un POKE 704,0:DOKE # A6,C000 marche. Le premier POKE est l'équivalent du GRAB. L'intérêt est qu'on peut l'employer n'importe quand, alors que GRAB doit être la première instruction d'un programme. Il ne faudra en revanche plus compter sur l'écran *(V. 59)*.

Enfin il y a la possibilité d'utiliser 16 K de RAM supplémentaires avec le KGB dont on peut se servir à partir du Basic. Ou, à l'aide d'une routine machine et d'un bit de commande (du CIA par exemple), on peut masquer la ROM à l'aide du signal MAP *(Voir le chapitre Extensions)*.

les fichiers machine

Ce n'est pas tout à fait du langage machine, ce sont des fichiers de variables non Basic. Ils seront situés en principe dans la RAM, protégée du Basic par HIMEM, et accessibles en écriture par POKE (ou DOKE), en lecture par PEEK (ou DEEK). Imaginons par exemple que nous voulions disposer d'un tableau de 100 valeurs de 0 à 99. Plutôt que faire un tableau classique du style :

```
DIM A%(9,9)
FOR U=0 TO 9
   FOR D=0 TO 9
      A%(D,U)=10×D+U
NEXT:NEXT
```

on ferait un fichier machine comme suit :

```
HIMEM 10000:HM=10000
FOR U=0 TO 9
   FOR D=0 TO 9
      POKE HM+10×D+U,10×D+U
NEXT:NEXT
```

et l'on récupèrerait ensuite nos valeurs par :

```
A = PEEK(HM+10×D+U)
```

Ce n'est guère plus compliqué qu'un tableau Basic, et ce fichier ne demande que 100 octets contre 209 pour le tableau correspondant. Ceci ne sera valable que pour stocker des entiers de 0 à 255, ne demandant qu'un octet, ou 8 bits. Si les variables s'expriment sur 16 bits, on peut préférer employer un tableau d'entiers plutôt qu'un fichier-machine, selon les cas. A noter que l'on a pas besoin de dimensionner un fichier-machine, et qu'il sera donc possible à tout moment de le modifier ou de l'éliminer sans toucher aux autres variables, alors qu'il faut jongler avec les pointeurs pour effacer un tableau donné sans toucher aux autres variables indicées ou non.

Mais bien souvent une variable ne va pas demander un nombre maximal de 8 ou 16 bits. On peut envisager de créer des fichiers-machine de groupes de 3 octets. Ce sera plus économique que d'employer des variables réelles, même si les manipulations ne sont pas faciles.

La variable pourra demander également moins de 8 bits, parfois un seul bit (un pixel sera allumé ou non sur l'écran par exemple). Dans ce cas il ne faudra plus manipuler des octets, mais des bits à l'intérieur des octets. Le Basic est assez mal équipé pour cela, bien que l'ORMOS dispose des essentielles fonctions logiques OR et AND. Par contre le langage machine possède des instructions très puissantes pour manipuler les octets bit par bit.

55 HARDCOPY d'écran HIRES rapide à partir du Basic

Une bonne méthode pour transférer un écran HIRES sur l'imprimante est de transmettre directement les octets contenant la mémoire-écran. Cette méthode rapide comporte néanmoins quelques inconvénients :

— le bit-map est codé sur les 6 bits de poids faible. Il faudra au moins tester le bit 6 pour déterminer si l'octet n'est pas un attribut,
— ce fait impose de n'utiliser que 6 aiguilles de l'imprimante, et de programmer un saut de ligne de 6 points. Or cette fonction n'est pas disponible sur toutes les machines,
— on recopie un écran ayant subi une rotation d'un quart de tour. De plus, selon la pondération des aiguilles de l'imprimante, les CHAR peuvent être imprimés à l'envers !
— enfin on recopie un écran de 240 x 200 points, ce qui ne représente souvent pas grand-chose par rapport à la résolution de l'imprimante. On aimerait pouvoir faire des agrandissements, des symétries sur les deux axes, mais c'est délicat en largeur.

La fonction POINT(X,Y) représente une bonne solution à tous ces problèmes, mais les auteurs de logiciels l'ont souvent jugée trop lente et ont préféré écrire sur Seikosha GP-100 (qui ne peut imprimer en mode graphique que des groupes de 7 points) des routines machine qui transforment 7 hexuplets en 6 septuplets. Ces routines sont évidemment très rapides et satisfaisantes tant qu'on les utilise telles quelles, mais toute modification ou adaptation pose d'énormes problèmes.

En bref, voici nos routines :

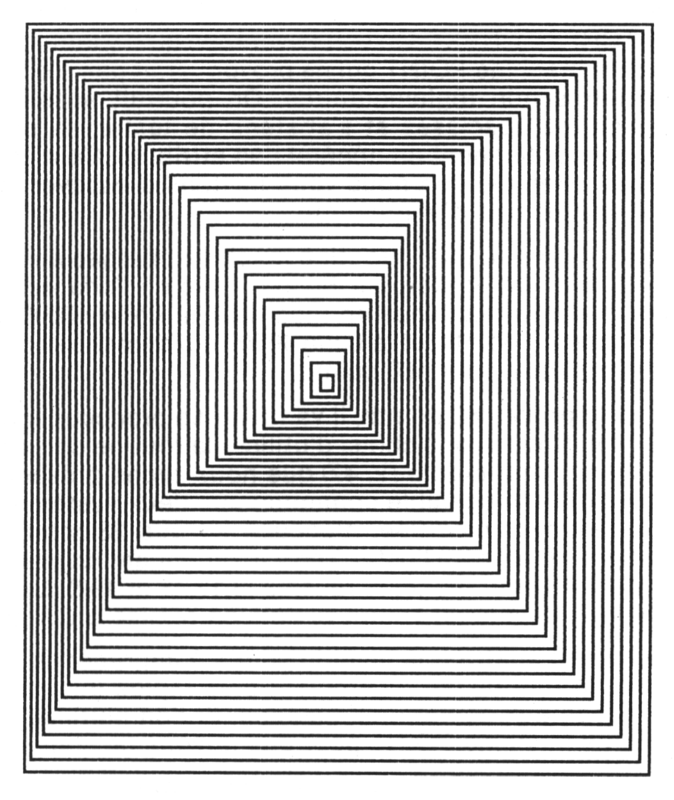
```
9 REM code machine pour 45000
10 A=1024
20 READA$:IFA$="ZZ"THEN100
30 B=VAL("#"+A$):POKEA,B:A=A+1:GOTO20
40 DATA A5,A8,F0,05,05,00,85,00,60
50 DATA A6,00,20,16,04,20,16,04,A9,00,85,00,60
60 DATA AD,0D,03,29,02,F0,F9,8E,01,03
70 DATA AD,00,03,A8,29,EF,8D,00,03,8C,00,03,60
80 DATA ZZ
```

```
99 REM magnifique exemple graphique
100 X1=239 : Y1=199
110 PAPER0:INK5
120 HIRES:PRINTCHR$(17)
130 FORA=0TO44STEP2
140 X0=A:Y0=A:CURSET X0,Y0,1
160 DRAW X1,0,1:DRAW 0,Y1,1
170 DRAW-X1,0,1 : DRAW 0,-Y1,1
200 X1=X1-6:Y1=Y1-5:NEXT
220 B=A+X1+2:C=A+Y1
230 FORA=0TO32STEP2
240 X0=B-A:Y0=C-A:CURSETX0,Y0,1
250 DRAW -X1,0,1:DRAW0,-Y1,1
260 DRAWX1,0,1:DRAW0,Y1,1
270 X1=X1-6:Y1=Y1-5:NEXT:END

29999 REM hardcopy SEIKOSHA GP-100
30000 POKE782,64
30010 POKE 49,255  '(ORIC)
30011 POKE 598,255 '(ATMOS)
30040 A=1:B=2:C=4:D=8:E=16:F=32:G=64:N=128
30050 FORY=0TO189STEP7
30060 H=Y+1:I=Y+2:J=Y+3:K=Y+4:L=Y+5:M=Y+6
30070 LPRINTCHR$(8)
31000 FORX=0TO239:Z=N
32001 IFPOINT(X,Y)THENZ=ZORA
32002 IFPOINT(X,H)THENZ=ZORB
32004 IFPOINT(X,I)THENZ=ZORC
32008 IFPOINT(X,J)THENZ=ZORD
32016 IFPOINT(X,K)THENZ=ZORE
32032 IFPOINT(X,L)THENZ=ZORF
32064 IFPOINT(X,M)THENZ=ZORG
32256 LPRINTCHR$(Z);:NEXT:NEXT
32500 POKE782,192:END

44998 REM hardcopy EPSON MX-82
44999 REM double largeur, double hauteur
45000 POKE782,64:POKE0,0:DOKE#2F5,1024
46000 LPRINTCHR$(27)"A"CHR$(8);
```

```
46500 A$=CHR$(27)+"K"+CHR$(224)+CHR$(1)
47000 FORY=0TO199STEP4
47100 C=Y+1:B=Y+2:A=Y+3
47200 LPRINTA$;
48000 FORX=0TO239
48131 IFPOINT(X,A)THEN!
48140 IFPOINT(X,B)THEN!
48176 IFPOINT(X,C)THEN!
48320 IFPOINT(X,Y)THEN!
48384 !:NEXT:LPRINT:NEXT
48500 POKE782,192
```



HARDCOPY d'écran, double largeur - double hauteur, obtenu au moyen de notre programme sur EPSON MX 82 (réduit à 59/100).

La routine Seikosha est entièrement Basic. On voit que toutes les constantes apparaissent sous forme de variables d'une seule lettre, qu'on a choisi de faire appel à l'opérateur OR plutôt qu'au + (et surtout au × et ˆ). Pour ces raisons une copie d'écran prend sept minutes environ.

Les numéros des lignes 32001 à 32256 ne sont pas choisis au hasard, 32000 étant multiple de 256, les constantes A,B,C... équivalent à l'octet de poids faible du numéro de ligne. En conséquence on pourrait faire dans chaque ligne Z = ZORPEEK(# A8), ou mieux appeler une routine machine qui ferait la même chose.

La routine telle qu'elle est présentée ne recopie que les 196 premières rangées de l'écran (Y = 0 à 195). Pour une recopie totale il faudrait réécrire les lignes 30060 à 32008 après 32256 et terminer par un LPRINTCHR$(Z);:NEXT. On voit l'avantage qu'il y aurait à passer par la routine machine POINT qui ne provoquerait pas d'erreur pour des paramètres non réglementaires. On pourrait aussi recopier l'écran selon les abscisses et s'abstenir de tester les premières colonnes qui contiennent en général des attributs. Ce n'est pas le cas de notre exemple que nous vous invitons à essayer même si vous ne disposez pas d'imprimante.

La seconde routine utilise la routine machine chargée en début de programme. Elle est écrite pour Epson, mais on l'adaptera facilement pour d'autres machines en modifiant la ligne 46000 qui fixe le saut de ligne à 8 points, et la ligne 46500 qui demande le passage en mode graphique pour 480 points.

Vu que l'on imprime 480 points, il va falloir se préoccuper du pointeur 48 *(V. 45)*. Nous avons choisi d'utiliser la routine de sortie d'un caractère vers l'imprimante *(V. 47)* que nous avons présentée plus haut, seule solution possible commune aux deux systèmes. Cette routine, légèrement modifiée, n'occupe ici que 23 octets (lignes 60-70). La ligne 40 prépare l'octet 0 en y additionnant les valeurs successives de # A8 pour les tests réussis (3,12,48 et 192). La valeur 0 provoque deux envois de l'octet 0, puis sa remise à 0 (ligne 50).

Bien que l'on ait quatre fois plus de points à imprimer que dans le dernier cas, la routine est légèrement plus rapide (moins de sept minutes), grâce aux opérations machine. Et l'on n'utilise toujours pas la routine machine POINT pour des raisons de compatibilité.

Pour une copie identique sur Seikosha, il faudrait faire deux passages du chariot par groupe de 7 Y, et tester deux fois le point milieu Y + 3. On aurait :

30040 A = 3:B = 12:C = 48:D1 = 64:D2 = 1:E = 6:F = 24:G = 96:N = 128

puis une première série de FOR X avec Z = ZORA,B,C,D1 et une deuxième avec Z = ZORD2,E,F,G. Pour éviter de s'occuper du pointeur 48, il faudrait expédier deux fois chaque caractère grâce à l'option de répétition de la GP—100 :

LPRINTCHR$(28)CHR$(2)CHR$(Z); : NEXT



ECRAN ET ROUTINES MACHINE

Nous présentons ici les adresses essentielles concernant l'écran ainsi que quelques utilitaires en LM permettant de le manipuler, entre autres utilisations.

56 Adresses écran

Le bit 0 de l'adresse #2C0 (704) indique le mode écran (0 pour TEXT, 1 pour HIRES). Le bit 1 est normalement à 1 et mis à 0 par un GRAB. Les autres bits de cet octet semblent inactifs. On a donc PEEK(704) = 2 (ou 0) en mode TEXT et 3 en HIRES.

Si la compatibilité Oric 1/Atmos est totale en HIRES, à l'exception du DRAW, 0,0 *(V. 40)*, il y a quelques différences en TEXT que nous présentons immédiatement :

— dans une instruction PLOT, la coordonnée X doit être augmentée de 1 sur l'Atmos ;
— l'Atmos imprime un espace avant un chiffre ; c'est en fait la conséquence de la correction de la fonction STR$, car PRINT A est équivalent à PRINT STR$(A) *(V. 36) ;*
— la gestion horizontale du curseur n'est pas la même en mode normal 38 colonnes. L'Oric se comporte toujours comme s'il était en mode 40 colonnes, alors que le curseur est parfaitement géré sur Atmos ;
— la gestion de la mémoire écran et les adresses concernées sont différentes. Sur Oric les adresses #26D-#D26E (621-622) contiennent l'adresse de début de la première ligne moins 40, l'adresse #26F (623) contient le nombre de lignes. Pour récupérer la ligne STATUS par exemple il faut faire :

DOKE 621,47960 : POKE 623,28

Sur Atmos, #27A-#27B (634-635) contiennent l'adresse de la première ligne, #278-#279 (632-633) l'adresse de la seconde ligne,

#27C-#27D (636-637) le nombre d'octets à translater, et #27E (638) le nombre de lignes. Pour récupérer la ligne STATUS sur Atmos, il faut faire :

DOKE 634,48000:DOKE 632,48040:DOKE 636,28*40:POKE 638,28

On peut à l'aide de ces adresses définir une fenêtre horizontale sur l'écran, le reste de celui-ci n'étant alors accessible que par des PLOT ou des POKE.

— l'Atmos possède aussi la possibilité de définir une fenêtre verticale grâce aux adresses 48-49 *(V. 46)*, sans équivalence directe sur l'Oric.

Ces divergences vues, revenons à notre ORMOS.

L'écran TEXT est constitué de 28 lignes de 40 colonnes. A chaque case correspond un octet de RAM. Il occupe donc 1120 octets d'adresses #BB80 à #BFDF (48000 à 49119). On peut poker dans une case soit un code ASCII supérieur à 31, et le caractère correspondant sera affiché, soit un attribut inférieur à 32. Dans ce cas rien ne sera affiché dans cette case, mais les suivantes de la même ligne seront affectées par cet attribut. Les attributs de 0 à 7 définissent la couleur de l'encre, de 16 à 23 la couleur du papier. Le système réserve en principe les deux premières cases de chaque ligne pour la définition du papier et de l'encre, mais on peut poker ce que l'on veut dans n'importe quelle case. On peut afficher des caractères dans les deux premières colonnes en mode 40 colonnes. On accède à ce mode en mettant à 1 le bit 5 de l'adresse #26A (618), par un POKE 618,PEEK(618) OR 32 ou un PRINT CHR$(29). A noter qu'en ce mode les caractères seront affichés en blanc sur fond noir. Il est néanmoins possible de les afficher en noir sur blanc en utilisant l'inversion vidéo. C'est le bit 7 de l'octet correspondant à une case de l'écran qui commande cette inversion. Un POKE 48040, 65 + 128 permet d'obtenir un A noir sur blanc bien que le papier de la ligne soit noir.

Le bit 0 de l'adresse #26A (toujours 618) commande l'affichage du curseur (ou plutôt son clignotement).

Les adresses #268-#269 (616 et 617) contiennent les positions verticale et horizontale du curseur. PEEK(617) correspond donc au POS(0). Par ailleurs les adresses #12-#13 (18-19) contiennent l'adresse de la case écran où se trouve le curseur. Il est ainsi possible de commander une impression par PRINT n'importe où sur l'écran en dokant en 18 l'adresse voulue.

Les caractères sont définis dans une matrice de 6 × 8. Les motifs standard sont contenus dans la table des caractères recopiée à par-

tir de la ROM de # B500 à # B7FF. Il est en fait plus simple de retenir pour adresse de départ # B400 (46080), ce qui permet de dénicher aisément les 8 octets définissant un ASCII A aux adresses # B400/Ax8 à # B407 + A58. Mais on ne peut pas redéfinir les ASCII 0 à 31 (ou 128 à 159), et la page # B4 est disponible pour l'utilisateur. Il en va de même pour la page # B8, et la table des motifs graphiques commence en # B900, bien que l'on prenne # B800 comme base de calcul. A noter que la table des motifs graphiques n'est pas recopiée à partir de la ROM, mais calculée selon le code binaire des caractères.

L'écran HIRES occupe 8 000 octets d'adresses # A000 à # BF (40960 à 48959), répartis en 200 rangées de 40 octets. Le bit 7 de chaque octet commande l'inversion vidéo. Le bit 6 définit si les bits suivants représentent un attribut ou le motif des 6 pixels correspondants sur l'écran. Lorsque ce bit est à 0 les attributs sont utilisables exactement comme en mode TEXT. De même, les deux premières colonnes (pixels 0 à 11 en abscisse) sont réservées aux couleurs papier et encre, mais en l'absence d'instructions PAPER ou INK après un HIRES, tous les octets de l'écran sont chargés avec la valeur 64 (bit 6 = 1), autorisant ainsi le graphisme en blanc sur noir sur tout l'écran (noir sur blanc en inversion vidéo). En mode TEXT on peut poker un attribut dans n'importe quelle case de l'écran, mais celle-ci sera perdue pour le graphisme.

Les adresses # 219- # 21A (537-538) contiennent les coordonnées horizontale et verticale du curseur (CURSET). La fonction DRAW demandant des coordonnées relatives, il sera souvent pratique de simuler une commande en coordonnées absolues en faisant :

DRAW X — PEEK(537), Y — PEEK(538), FB

Avec FB = 1, on a l'équivalent de DRAW X1-X0,Y1-Y0,1 et avec FB = 3 l'équivalent de CURSET X,Y,3.

Enfin, les trois lignes de TEXT en bas d'écran occupent les adresses # BF40 à # BFDF 48960 à 49119). Il est éventuellement possible de faire du graphisme sur la totalité de l'écran en pokant dans chacune de ces cases un code ASCII différent. On programmera le graphisme en modifiant les tables de caractères standard et/ou alternés.

Dernière curiosité : un POKE 48000, 31 suivi d'un POKE 704,3 donnera 128 lignes en mode HIRES et 12 lignes en mode TEXT.

57 Inversion vidéo

L'inversion vidéo présente la particularité de permettre de changer couleur et fond d'une case sans perdre les deux cases précédentes pour l'affichage. On l'obtient en Basic par la fonction PLOT CHR$(A) avec A > 127 ou en pokant directement une valeur supérieure à 127 dans une case-écran. A noter qu'un CHR$(128) sera affiché alors qu'un CHR$(0) sera ignoré.

L'inversion transformera donc les deux couleurs de fond et d'encre en leurs couleurs complémentaires, le noir en blanc, le bleu en jaune, etc.

La routine que voici inverse l'état du bit 7 par la très intéressante instruction EOR. On opère sur les adresses # A000 à # BFDF, c'est-à-dire que l'appel de la routine provoquera l'inversion aussi bien d'un écran TEXT que HIRES, un nouvel appel rétablissant l'écran original, ou presque.

```
         I 0400 - 0428
0400:    A0  00          LDY #$00
0402:    84  00          STY $00
0404:    A9  A0          LDA #$A0
0406:    85  01          STA $01
0408:    A9  E0          LDA #$E0
040A:    85  02          STA $02
040C:    A9  BF          LDA #$BF
040E:    85  03          STA $03
0410:    B1  00          LDA ($00),Y
0412:    49  80          EOR #$80
0414:    91  00          STA ($00),Y
0416:    E6  00          INC $00
0418:    D0  02          BNE $041C
041A:    E6  01          INC $01
041C:    A5  00          LDA $00
041E:    C5  02          CMP $02
0420:    D0  EE          BNE $0410
0422:    A5  01          LDA $01
0424:    C5  03          CMP $03
0426:    D0  E8          BNE $0410
0428:    60              RTS
```

Il est possible de changer les adresses de départ et d'arrivée. On peut aussi supprimer les lignes 0402 à 040E et doker en 0 et 2 ces adresses de départ et d'arrivée avant d'appeler la routine.

Il est possible de fournir une autre valeur à l'EOR en 0412, ce qui peut provoquer des choses tout à fait bizarres. Eviter toutefois l'inversion du bit 6 en HIRES.

58 Charger une zone mémoire avec une valeur en donnée

Nous ne présentons que le chargeur Basic de la routine, celle-ci étant pratiquement identique à la routine précédente.

```
9 REM Mise à V d'une zone mémoire A-E
10 A=1024
20 READ A$:IF A$="ZZ"THEN 100
30 B=VAL("#"+A$):POKE A,B
40 A=A+1:GOTO 20
50 DATA A0,00,A5,04,91,00
60 DATA E6,00,D0,02,E6,01
70 DATA A6,00,E4,02,D0,F2
80 DATA A6,01,E4,03,D0,EC
90 DATA 60,ZZ
100 INPUT "Début";A
110 INPUT "Fin";E
120 INPUT "Valeur";V
130 DOKE 0,A:DOKE 2,E:POKE4,V
140 CALL #400:STOP
```

Ce seront les octets d'adresses A à E-1 qui seront chargés avec la valeur V, ceci à dessein car on donnera souvent à E une valeur connue, comme le début de la table de caractères, dont on ne désirera pas voir le contenu modifié.

Cette routine peut servir à effacer une portion d'écran TEXT (V = 32) ou HIRES (V = 64), ou encore à préparer une zone mémoire destinée à du graphisme sur imprimante.

A noter qu'il y aurait une solution Basic rapide pour V = 0. Il suffirait de charger les pointeurs DT et FT avec A-7, DC et HM avec E,

et de dimensionner un tableau avec un nombre d'éléments correspondant à E-A divisé par 2, 3 ou 5 selon la nature du tableau...

59 Curieuse utilisation du FRE(0)

Entrez donc le petit programme suivant :

```
0 REM Ecran et variables chaînes
10 GRAB : DOKE #A6,#BFE0 : DOKE #A2,#B500
20 CLS : PRINT"Veuillez entrer quelques mots...
30 FOR A=10 TO 0 STEP -1
40 INPUT A$(A) : A$(A)=A$(A)+" "
50 NEXT
60 CLS : A=FRE(0)
```

Curieux, non ? Veuillez revoir en cas de besoin le chapitre sur les pointeurs. On pourrait se servir de cette idée pour changer très rapidement d'écran, en mode TEXT ou HIRES. Il suffirait en mode TEXT de 5 variables chaînes pour définir tout un écran, ou plus commodément 7 variables de 160 caractères correspondant chacune à 4 lignes d'écran. Il serait possible de définir tranquillement ces variables tant que le pointeur DC se trouverait en dessous de la table de caractères (DOKE # A2,# B500), puis d'obtenir quasi instantanément un nouvel écran par un FRE(0). Il serait encore possible de ne pas agir directement sur l'écran, mais sur la table de caractères, en positionnant HM en # B800 OU # BB00.

Cette méthode pourrait être une bonne solution pour pallier la lenteur du scrolling de l'ORMOS, ou pour d'autres applications. Il faudrait bien sûr jongler avec les pointeurs DC et HM sans arrêt, pour éviter d'altérer inconsidérément les tables de caractères. A noter que l'on peut réinitialiser ces tables en appelant # F89B sur Oric et # F8D0 sur Atmos.

60 Transférer une zone mémoire

C'est encore une fonction très classique, qui possède de nombreuses applications. Pour appeler la routine, on fera :

DOKE 0,A : DOKE 2,E :DOKE 4,T : CALL 1024

avec A = adresse du premier octet à transférer,

E = adresse du dernier octet à transférer,
T = adresse de destination du premier octet.

La particularité de cette routine réside en ce que E peut être inférieur à A. Dans ce cas, (A) sera transféré en T, (A-1) en T-1, etc.

La routine est présentée en deux parties, la seconde correspondant à ce cas précis. Si l'on a toujours E>A, on peut omettre la seconde partie.

```
I400-444
0400:   38          SEC
0401:   A5  02      LDA $02
0403:   E5  00      SBC $00
0405:   85  02      STA $02
0407:   A5  03      LDA $03
0409:   E5  01      SBC $01
040B:   90  38      BCC $0445
040D:   AA          TAX
040E:   A0  00      LDY #$00
0410:   B1  00      LDA ($00),Y
0412:   91  04      STA ($04),Y
0414:   C8          INY
0415:   C4  02      CPY $02
0417:   D0  F7      BNE $0410
0419:   18          CLC
041A:   98          TYA
041B:   65  00      ADC $00
041D:   85  00      STA $00
041F:   A5  01      LDA $01
0421:   69  00      ADC #$00
0423:   85  01      STA $01
0425:   98          TYA
0426:   65  04      ADC $04
0428:   85  04      STA $04
042A:   A5  05      LDA $05
042C:   69  00      ADC #$00
042E:   85  05      STA $05
0430:   8A          TXA
0431:   D0  01      BNE $0434
0433:   60          RTS
```

```
0434:    A0  00       LDY #$00
0436:    B1  00       LDA ($00),Y
0438:    91  04       STA ($04),Y
043A:    C8           INY
043B:    D0  F9       BNE $0436
043D:    E6  01       INC $01
043F:    E6  05       INC $05
0441:    CA           DEX
0442:    D0  F2       BNE $0436
0444:    60           RTS
```

```
I445-476
0445:    AA           TAX
0446:    CA           DEX
0447:    38           SEC
0448:    A5  00       LDA $00
044A:    E5  02       SBC $02
044C:    85  00       STA $00
044E:    A5  01       LDA $01
0450:    E9  00       SBC #$00
0452:    85  01       STA $01
0454:    A5  04       LDA $04
0456:    E5  02       SBC $02
0458:    85  04       STA $04
045A:    A5  05       LDA $05
045C:    E9  00       SBC #$00
045E:    85  05       STA $05
0460:    A4  02       LDY $02
0462:    B1  00       LDA ($00),Y
0464:    91  04       STA ($04),Y
0466:    88           DEY
0467:    D0  F9       BNE $0462
0469:    E8           INX
046A:    F0  06       BEQ $0472
046C:    C6  01       DEC $01
046E:    C6  05       DEC $05
0470:    D0  F0       BNE $0462
```

```
0472:    B1  00        LDA ($00),Y
0474:    91  04        STA ($04),Y
0476:    60            RTS
```

Cette routine a été conçue dans un souci de rapidité optimale. Elle peut en théorie transférer 64 K en une seconde. Du fait de cette rapidité elle occupe plus de place qu'il ne serait nécessaire en employant une technique d'incrémentation similaire à celle des routines précédentes. On s'est débrouillé ici pour manipuler les données par blocs de 256 octets.

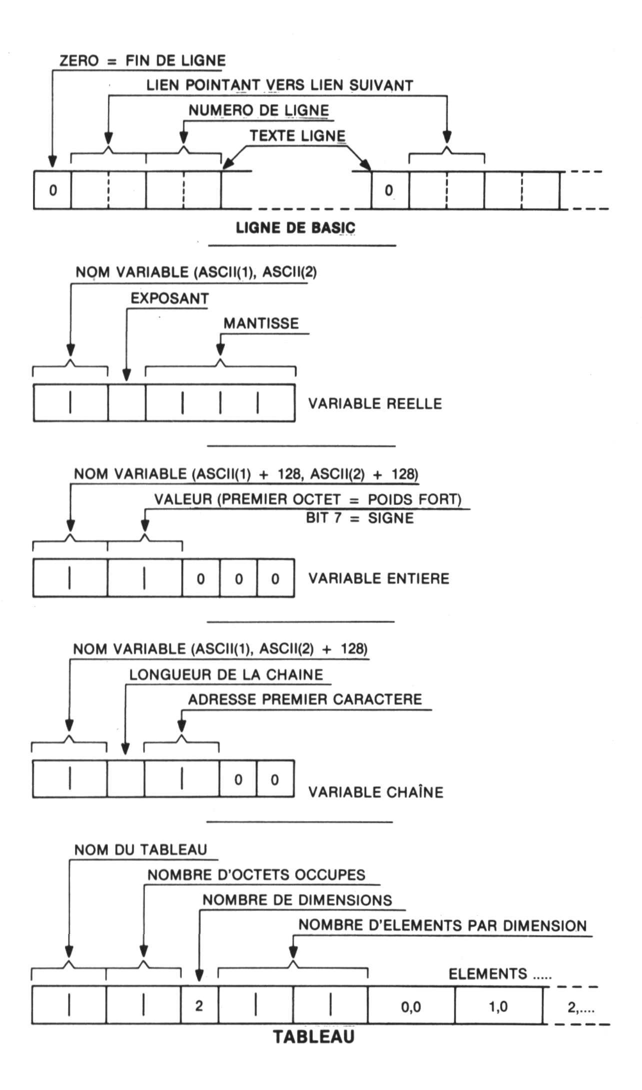
Cette routine a été conçue pour être utilisée dans un programme de traitement de texte, où l'on a besoin de manipuler des blocs parfois importants de données. Elle peut toutefois recevoir beaucoup d'autres applications.

La plus évidente est le transfert d'écran. On peut ainsi stocker plusieurs écrans TEXT ou HIRES en RAM et passer très rapidement de l'un à l'autre. Ceci permet par exemple de sauvegarder un écran HIRES sans défaut sur Oric.

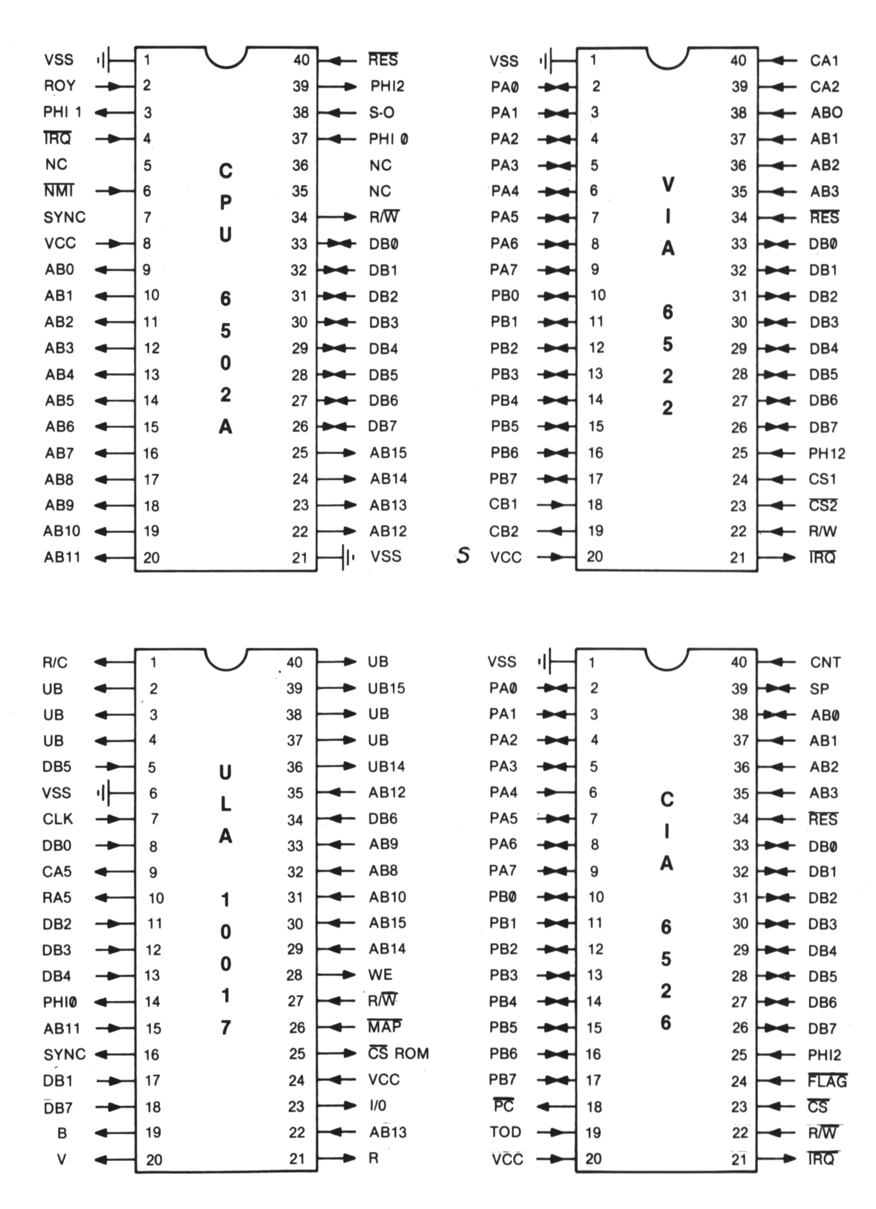
Nous allons conclure par un exemple d'animation graphique sans prétention. On remarque qu'il est très facile d'écrire les lignes 140 à 230 également en LM. Songez que ce programme transfère 200 fois 16 000 octets, soit plus de trois méga-octets...

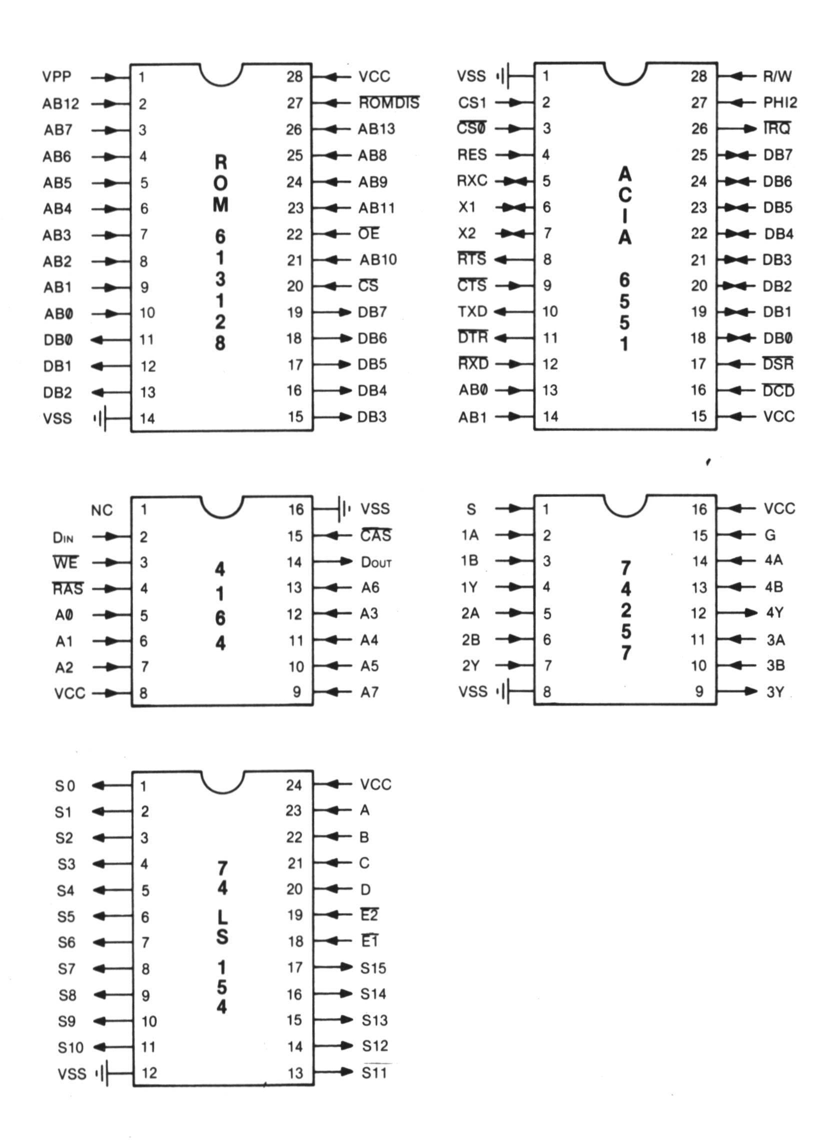
```
99 REM Coucher de soleil au Japon
100 PAPER4:HIRES:PRINTCHR$(17)
110 PAPER3:INK1
120 CURSET120,100,1
130 FORI=2TO95:CIRCLEI,1:NEXT
140 FORA=1TO199
150 DOKE0,#A000
160 DOKE2,#BF3F
170 DOKE4,#4000
180 CALL1024
190 DOKE0,#4000
200 DOKE2,#5F17
210 DOKE4,#A028
220 CALL1024
230 NEXT
```

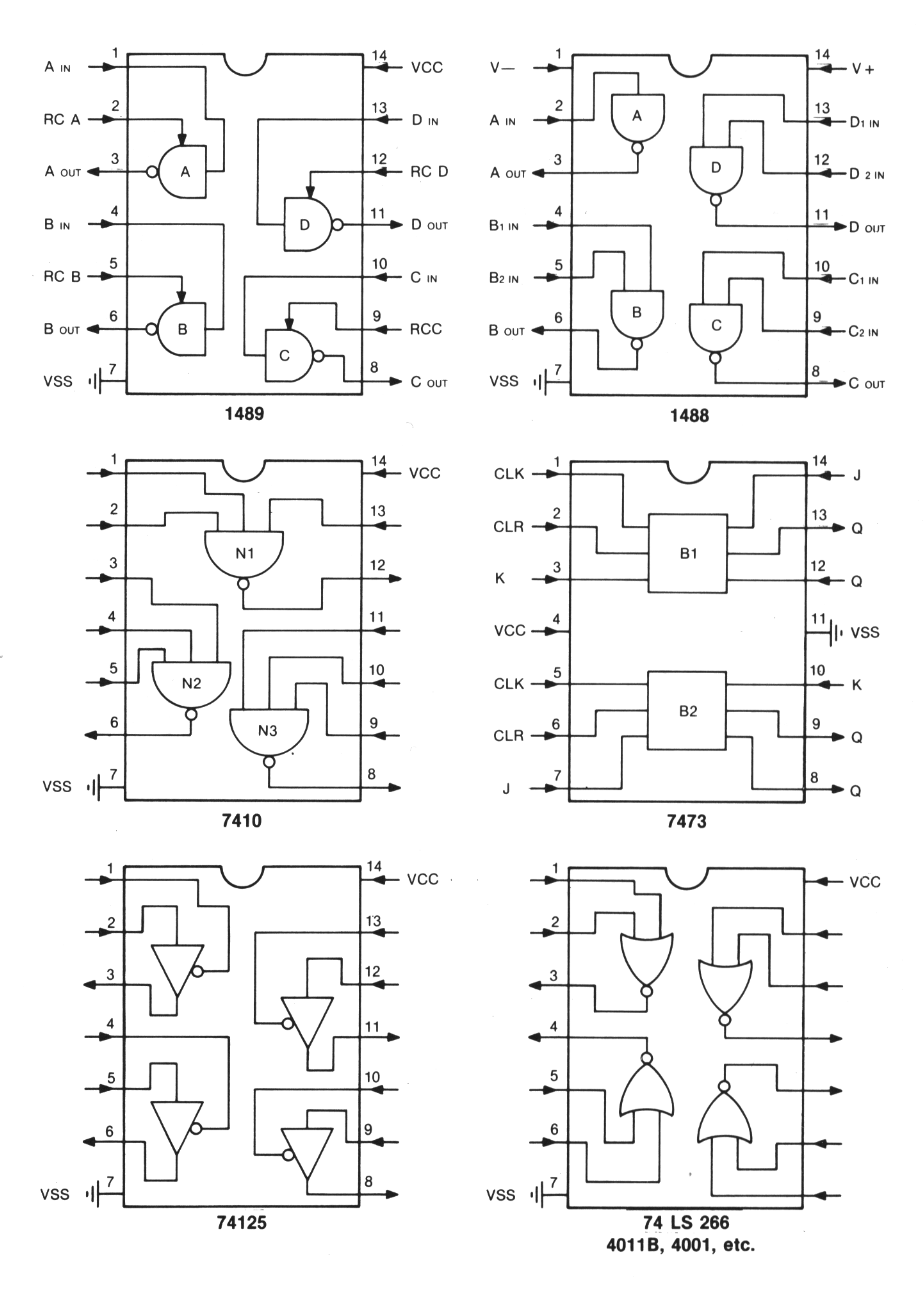
Carte de la RAM de l'ORMOS



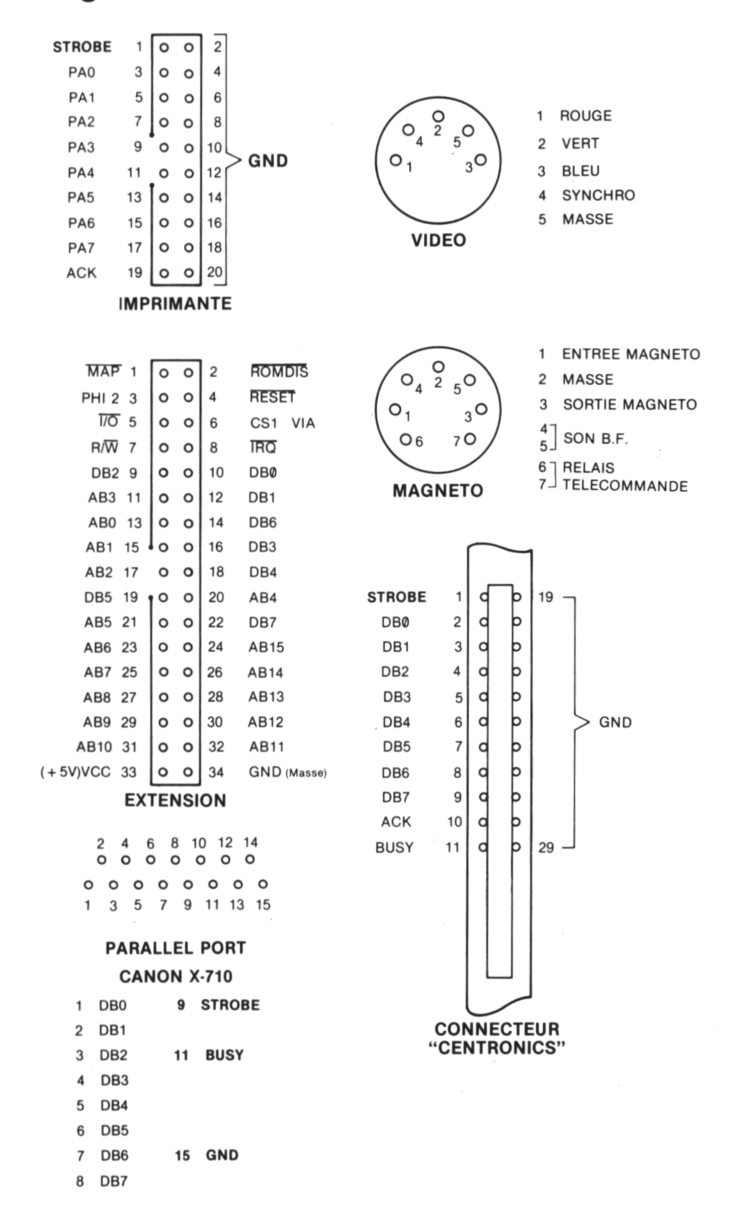
Brochages de CI



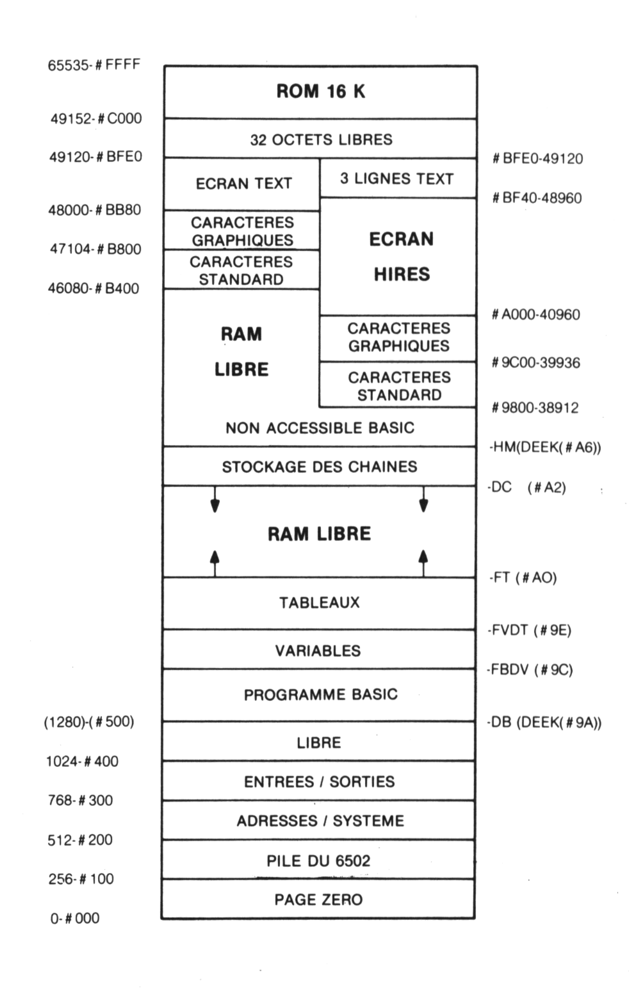

ROM 613128
VPP 1
AB12 2
AB7 3
AB6 4
AB5 5
AB4 6
AB3 7
AB2 8
AB1 9
AB0 10
DB0 11
DB1 12
DB2 13
VSS 14
28 VCC
27 ROMDIS
26 AB13
25 AB8
24 AB9
23 AB11
22 OE
21 AB10
20 CS
19 DB7
18 DB6
17 DB5
16 DB4
15 DB3
ACIA 6551
VSS 1
CS1 2
CS0 3
RES 4
RXC 5
X1 6
X2 7
RTS 8
CTS 9
TXD 10
DTR 11
RXD 12
AB0 13
AB1 14
28 R/W
27 PHI2
26 IRQ
25 DB7
24 DB6
23 DB5
22 DB4
21 DB3
20 DB2
19 DB1
18 DB0
17 DSR
16 DCD
15 VCC
4164
NC 1
DIN 2
WE 3
RAS 4
A0 5
A1 6
A2 7
VCC 8
16 VSS
15 CAS
14 DOUT
13 A6
12 A3
11 A4
10 A5
9 A7
74257
S 1
1A 2
1B 3
1Y 4
2A 5
2B 6
2Y 7
VSS 8
16 VCC
15 G
14 4A
13 4B
12 4Y
11 3A
10 3B
9 3Y
74LS154
S0 1
S1 2
S2 3
S3 4
S4 5
S5 6
S6 7
S7 8
S8 9
S9 10
S10 11
VSS 12
24 VCC
23 A
22 B
21 C
20 D
19 E2
18 E1
17 S15
16 S14
15 S13
14 S12
13 S11


A IN
RC A
A OUT
B IN
RC B
B OUT
VSS
VCC
D IN
RC D
D OUT
C IN
RCC
C OUT
1489
V—
A IN
A OUT
B1 IN
B2 IN
B OUT
VSS
V +
D1 IN
D 2 IN
D OUT
C1 IN
C2 IN
C OUT
1488
N1
N2
N3
VSS
VCC
7410
CLK
CLR
K
VCC
CLK
CLR
J
B1
B2
J
Q
Q
VSS
K
Q
Q
7473
VSS
VCC
74125
VCC
VSS
74 LS 266
4011B, 4001, etc.

Brochages des connecteurs



Représentation des divers éléments en RAM

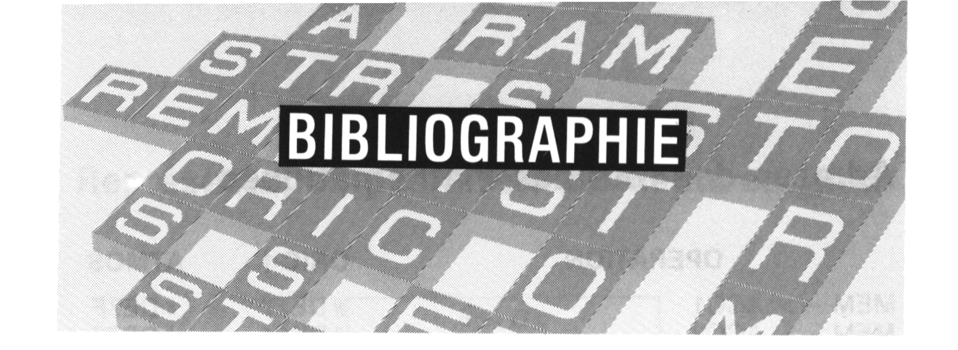


Adresse des routines mathématiques

OPERATION	ORIC-1	ATMOS
MEM ---> ACC1	# DE77	# DE7F
MEM ---> ACC2	# DD51	# DD55
ACC1 ---> MEM	# DEAC	# DEB4
ACC1 ---> ACC2	# DEDD	# DEE5
ACC2 ---> ACC1	# DECD	# DED5
ACC1 ---> Entier en # D3-D4	# DF74	# DF8C
Entier en # D1-D2 ---> ACC1	# D3F5	# D4A1
ACC1 ---> chaîne ASCIIs débutant en # 100	# E0D1	# E0D5
ACC2 + ACC1 ---> ACC1	# DA9F	# DB2A
ACC2 — ACC1 ---> ACC1	# DA83	# DB0E
ACC2 x ACC1 ---> ACC1	# DCBF	# DCF5
ACC2 / ACC1 ---> ACC1	# DDE5	# DDE9
ACC2 ^ ACC1 --->ACC1	# E236	# E23A
ABS(ACC1) ---> ACC1	# DF31	# DF49
INT(ACC1) --->ACC1	# DFA5	# DFBD
SGN(ACC1) ---> ACC1	# DF12	# DF21
PI ---> ACC1	# D8EE	# DE77
COS(ACC1) ---> ACC1	# E387	# E38B
SIN(ACC1) ---> ACC1	# E38E	# E392
TAN(ACC1) ---> ACC1	# E3D7	# E3DB
ATN(ACC1) ---> ACC1	# E43B	# E43F
LN(ACC1) ---> ACC1	# DC79	# DCAF
EXP(ACC1) ---> ACC1	# E2A6	# E2AA
LOG(ACC1) ---> ACC1	# DDD0	# DDD4
SQR(ACC1) ---> ACC1	# E22A	# E22E

ACC1 : # D0 à # D5
ACC2 : # D8 à # DD
MEM : variable réelle sur 5 octets dont le premier est en DEEK (# 91)
Entier : nombre signé sur 16 bits correspondant à variable entière (%)

Routines mathématiques : nous donnons ici des points d'entrée légèrement différents de ceux déjà publiés.



BIBLIOGRAPHIE

Langage machine 6502

6502, programmation en langage assembleur, par L. A. Leventhal, éd. Radio.

S'il ne faut avoir qu'un livre, c'est celui-là. Ouvrage très complet dont le prix un peu élevé est justifié.

Programmation du 6502, par R. Zaks, éd. Sybex.

Très bon ouvrage d'une excellente présentation pédagogique.

L'Assembleur facile du 6502, par F. Monteil, éd. Eyrolles.

Un bon ouvrage d'introduction au LM, mais qui risque d'être insuffisant.

Carte de référence 6502, par F. Monteil, éd. Eyrolles.

Très bon outil pour qui n'a pas de logiciel d'assemblage.

Entrées-sorties

6502, programmation en langage assembleur, par L.A. Leventhal, éd. Radio.

Toujours lui ; description du 6522 et du 6551, et exemples de programmation de ces composants.

Applications du 6502, par R. Zaks, éd. Sybex.

Ne décrit que les coupleurs parallèles (VIA, PIA).

VIA 6522, éditions Publitronic.

Tout sur ce discret entremetteur.

Programmation en Assembleur 6809, par Bui Minh Duc, éd. Eyrolles.

Il s'agit bien sûr de 6809, mais cet ouvrage est l'un des seuls à présenter de façon claire toutes les faces de l'interface RS-232. Des programmes d'application sont donnés, plusieurs protocoles sont suffisamment commentés pour permettre une adaptation aisée en 6502.

Using the 64, par P. Gerrard, éd. Duckworth.

Le C64 utilise deux 6526. On trouve dans cet ouvrage la description détaillée de ce composant (en anglais).

Mastering the C64, par Jones et Carpenter, éd. Ellis Horwood Limited.

Toujours de l'anglais et du C64, mais des routines utilisant le 6526 facilement adaptables.

La conduite du C64, tome 2, par F. Monteil, éd. Eyrolles.

Description du 6526 en français.

ORMOS

Clefs pour Oric, par E. Flesselles, éd. PSI.

Beaucoup de choses intéressantes, de correspondances Oric-Atmos, mais attention au chapitre jeu d'instructions du 6502 (syntaxe déconcertante, erreurs).

Au Cœur de l'Atmos, par G. Bertin, éd. Inform'atic.

Trucs et adresses. Listing de la ROM de l'Atmos, malheureusement sans commentaires.

The Oric 1 Companion, par B. Maunder, éd. Linsac.

Listing de la ROM 1-0, toujours sans commentaires.

Divers

Expériences de logique digitale, par F. Huré, éd. ETSF.

Bon ouvrage d'initiation.

Programmation interne du C64, par M. Bathurst, éd. Datacap.

Un listing complet de la ROM du C64, avec chaque instruction commentée, toutes les variables système avec toutes les routines les appelant, toutes les sous-routines. Nous préférerions présenter un ouvrage similaire sur l'ORMOS, mais rappelons que le C64 est très proche de celui-ci, Microsoft oblige, et que nombre de routines sont exactement semblables, instruction par instruction. Les localisations des routines et des variables sont bien sûr différentes, mais on peut tirer beaucoup de cet ouvrage.

Revues

Micro-systèmes présente notamment des programmes en LM intéressants.

Micro et Robots publie des extensions pour systèmes à base de 6502 et autres.

Elektor offre des réalisations à base de 6502.

Radio-plans donne des utilitaires pour Oric.

La Commode fournit de bons renseignements sur les systèmes à base de 6502 (Commodore et ORMOS).

Logiciels

« Moniteur 1-0 » de Loriciels.

Sans doute un peu encombrant par rapport à ses fonctions, mais pas mal quand même.

« Assembleur 6502 » de La Commode.

« Basic Plus ».

Permet de doter l'Oric de trois fonctions très puissantes, MERGE, DELETE et RENUM.

COLLECTION POCHE informatique

1 - G. ISABEL, *50 programmes pour ZX 81*
2 - P. GUEULLE, *Montages périphériques pour ZX 81*
3 - C. GALAIS, *Passeport pour Applesoft*
4 - R. BUSCH, *Passeport pour Basic*
5 - M. ROUSSELET, *Mathématiques sur ZX 81*
6 - C. GALAIS, *Passeport pour ZX 81*
7 - G. PROBST, *50 programmes pour Casio FX-702 et 801 P*
8 - G. PROBST, *60 programmes pour Casio PB-100*
9 - M. SAAL, *Utilitaires pour ZX 81*
10 - C. GALAIS, *Passeport pour Commodore 64*
11 - D. RANC, *Assembleur du TRS 80*
12 - D. LASSERAN, *30 programmes pour Commodore 64*
13 - G. ISABEL, *Du ZX 81 au Spectrum, 25 programmes*
14 - P. MELUSSON, *Initiation à la micro-informatique, le microprocesseur*
15 - G. PROBST, *40 programmes pour Casio PB-700*
16 - C. GALAIS, *Passeport pour Basic TO 7 et TO 7-70*
17 - D. LASSERAN, *35 programmes pour Oric 1 et Atmos*
18 - G. PROBST, *40 programmes pour Canon X-07*
19 - P. MANGIN, *Jeu sur Commodore 64*
20 - G. ISABEL et B. NGUYEN VAN TINH, *Langage machine sur ZX 81*

Achevé d'imprimer
sur les presses
de l'Imprimerie Marcel Bon
70001 VESOUL
Dépôt légal : mars 1985
N° d'éditeur : 456
N° d'imprimeur : 2892

60 SOLUTIONS POUR ORIC 1 ET ATMOS

Cet ouvrage est un recueil d'idées, d'astuces tant logicielles que matérielles. Tout possesseur d'Oric 1 ou d'Atmos y trouvera de quoi **améliorer le fonctionnement ou les performances de sa machine,** de quoi **perfectionner sa programmation.**

Grâce à la présentation en sections claires et concises qu'a adoptée l'auteur, c'est un livre de consultation aisée et rapide qui vous permettra d'aller plus loin avec votre Oric.

Principaux chapitres :

- Architecture du système
- Problèmes matériels
- Interfaces
- Extensions
- Programme Basic en RAM
- Applications
- Langage machine
- Problèmes usuels Basic
- Imprimante
- Optimisation des programmes
- Ecran et routines machine.

ISBN 2-85535-104-9

TARIF CODE P


21
60 SOLUTIONS POUR ORIC
R. SCHULZ
ETSF